Escuela Politécnica Superior

Sistema de reconstrucción 3D del cuerpo humano a partir de múltiples vistas RGB-D

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Juan Miguel Castillo Zaragoza

Tutor/es:
Jorge Azorín López
Andrés Fuster Guilló

Junio 2019

Universitat d'Alacant
Universidad de Alicante

Universidad de Alicante

Departamento de Tecnología Informática y Computación

Trabajo de Fin de Grado

Sistema de reconstrucción 3D del cuerpo humano a partir de múltiples vistas RGB-D

Juan Miguel Castillo Zaragoza

Tutores

Dr. Jorge Azorín López

Dr. Andrés Fuster Guilló

Memoria presentada para aspirar al grado de:

GRADUADO EN INGENIERÍA INFORMÁTICA

Alicante, 15 de Junio de 2019

"La muerte no nos roba los seres amados. Al contrario, nos los guarda y nos los inmortaliza en el recuerdo."

François Mauriac

AGRADECIMIENTOS

La primera de estas líneas no podría ir dirigida hacia otra persona que no fuera ella, mi mujer, Natalia. Eternamente agradecido por esa comprensión y ese sustento infinitos e incondicionales que he recibido por tu parte a lo largo de estos años, en los buenos y en los malos momentos, y sin los cuales, nada de esto habría sido posible.

El siguiente párrafo como tampoco podría ser de otra manera, reservado a mis padres y hermano, por haber hecho posible que mi trayectoria hiciese paso por esta etapa, y una vez en ella, haberme dado todo su apoyo.

De estos cuatro años me llevo muchas amistades, pero en especial las de mis compañeros de "batalla" Luis y Daniel. Dicen que trabajando en equipo se llega más lejos y este ha sido el claro ejemplo de que compartiendo el camino con gente grande, el límite no es el cielo.

También merecen especial mención mis amig@s, que aún cuando he tenido que sacrificar el tiempo que les dedicaba, siempre han estado ahí, dándome fuerzas para seguir, celebrando mis logros y motivándome en los tropiezos.

Por último y no menos importante, a todos los profesionales que han dedicado el bien más preciado de sus vidas a transmitirme sus conocimientos, en especial a mis compañeros de equipo Jorge, Andrés, Marcelo y Víctor.

Por último gracias a todos los que han hecho esto posible y han quedado fuera de estas líneas y en especial, a ti que tienes este trabajo en tus manos.

RESUMEN

El presente trabajo aborda el proceso de adquisición 3D mediante cámaras RGB-D de objetos, la obtención de un modelo 3D y su representación texturizada para su posterior visualización mediante técnicas de realidad virtual.

Obtener modelos virtuales a partir de objetos reales es un problema ampliamente estudiado y que se ha afrontado desde distintos enfoques a lo largo de los años. El pipeline clásico incluye metodologías para resolver problemas como: la adquisición de múltiples vistas del objeto, el alineamiento de las nubes de puntos correspondientes a las diferentes vistas, la generación del modelo 3D a partir de las nubes de puntos y la proyección de las imágenes obtenidas desde las diferentes vistas para obtener el modelo texturizado.

Este trabajo que tienes entre tus manos nace estrechamente ligado a un proyecto de investigación denominado "*Modelado y visualización 4D del cuerpo humano para la mejora de la adherencia al tratamiento dietético-nutricional de la obesidad.*" con referencia "TIN2017-89069-R" perteneciente al programa Retos 2017. El principal objetivo de este proyecto de investigación es la especificación de un modelo de representación visual 4D (3D + Tiempo) del cuerpo humano para el análisis de la evolución morfológica ocasionada por cambios debidos al tratamiento de la obesidad. El modelo 3D se utilizará como núcleo para el desarrollo de un sistema de visualización del cuerpo que mejore el tratamiento de la obesidad. Se pretende proporcionar visualizaciones mediante realidad virtual inmersiva y simulaciones de la evolución consecuencia del tratamiento. Este proceso se lleva a cabo con la intención de estudiar su repercusión en la adherencia del paciente al tratamiento.

En este contexto el presente trabajo aborda la adquisición 3D del cuerpo humano a partir de redes de cámaras RGB-D, la obtención del modelo 3D del cuerpo, su representación texturizada y la exportación del modelo de forma que pueda observarse mediante realidad virtual. Para ello se ha desarrollado un sistema con sucesivas fases de procesado, desde la adquisición, modelado, hasta el texturizado. Las diferentes fases implementan métodos y utilizan herramientas que han sido cuidadosamente seleccionadas y adaptadas para conseguir los resultados esperados.

Espero disfrutes de él tanto como lo he hecho yo durante su desarrollo.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En este capítulo se expone el marco en el que se ha realizado el proyecto, su motivación y puesta en contexto. Además, se presenta un estado del arte en el que se describen las principales metodologías empleadas actualmente para abordar problemas del mismo tipo. Por último, se plantean los objetivos generales del proyecto y su planificación.

1.1 Motivación y contexto

Siempre he sido un amante de la electrónica y me formé en este campo porque era una de mis grandes pasiones. Aprendí a programar en Visual Basic con 11 años allá por 1995 y aunque la informática ha sido algo que siempre ha estado presente en mi vida, hasta ahora solo era un hobby. Por circunstancias de la vida y debido a mi ambición emprendedora esta pasión se orientó hacia la explotación de diversas oportunidades de negocio. Hace cuatro años me propuse dar un giro y me embarqué en la aventura de cursar el Grado de Ingeniería Informática. Durante estos cuatro años he aprendido infinidad de conceptos y he adquirido muchas habilidades gracias a trabajar junto a grandes compañeros y, sobre todo, a los profesores que he tenido.

Hace un año Jorge Azorín y Andrés Fuster me ofrecieron la oportunidad de colaborar en su grupo de investigación en el área de visión por computador lo que

acepté con mucha ilusión. En este grupo he tenido el placer de trabajar con grandes profesionales y compañeros como Marcelo Saval o Víctor Villena.

Ya asentado en el equipo y buscando alinearme con el trabajo realizado analicé su trayectoria que fue iniciada por Andrés con su tesis "Modelado de sistemas para visión realista en condiciones adversas y escenas sin estructura" (Fuster Guilló, 2004) en la que proponía soluciones para aplicar a diferentes escenarios en los que la imagen es adquirida en condiciones adversas de percepción, como pueden ser objetos que aparezcan en penumbra o deslumbrados.

Posteriormente, Jorge Azorín continuó trabajando con los inconvenientes de la visión en situaciones adversas y desarrolló en su tesis "Modelado de sistemas para visión de objetos especulares. Inspección visual automática en producción industrial" (Azorín López, 2008) diversas propuestas para mejorar la inspección visual en escenas de especularidad. Estudió los criterios que debe cumplir un sistema que afronte estas situaciones y desarrolló técnicas basadas en proyecciones de luz para resaltar defectos.

Marcelo Saval continuó con la línea de trabajo del equipo y la evolucionó al paso que también lo hacía la tecnología. Cuando elaboró su tesis, los sistemas de visión artificial ya giraban en torno a una percepción multidimensional del entorno y empezaban a haber sistemas más accesibles a nivel económico que te permitían llevar a cabo visión 3D y junto con ellos dispositivos de captura que permitían añadir una tercera dimensión, la de profundidad a las imágenes 2D, este tipo de sensores se clasifican como sensores RGB-D (Red Green Blue Depth). En su tesis "Methodology based on registration techniques for representing subjects and their deformations acquired from general purpose 3D sensors" (Saval-Calvo, 2015), Marcelo propone un conjunto de métodos con los que pretende mejorar la adquisición y el registro de datos 3D obtenidos a través de sensores RGB-D en situaciones no óptimas, desarrollando los procesos de adquisición en los límites de sensibilidad de este tipo de sensores.

Por último, Victor Villena se encuentra actualmente terminando su tesis enfocada a llevar a cabo una nueva propuesta para la obtención del registro deformable a partir de datos de una menor calidad como los provistos a partir de sensores RGB-D. Aunque ya hay estudios en base a este tema, está orientado a llevarlo a cabo con una mayor precisión que la obtenida hasta ahora. También Victor Villena, en su trabajo de fin de grado titulado "Análisis comparativo de métodos de calibrado para sensores RGB-D y su influencia en el registro de múltiples vistas"(Villena Martínez, 2015) hizo un análisis exhaustivo en el que se testeaban distintos métodos de calibrado para este tipo de sensores y se comparaban resultados de un proceso llevado a cabo en un sistema de registro global de las diferentes vistas.

La propuesta de mi incorporación al equipo surge a raíz de la necesidad de desarrollar labores para un proyecto de investigación subvencionado con fondos europeos denominado "*Modelado y visualización 4D del cuerpo humano para la mejora de la adherencia al tratamiento dietético-nutricional de la obesidad.*" con referencia "TIN2017-89069-R" perteneciente al programa Retos 2017 en el que Jorge Azorín es el investigador principal y que busca evolucionar y mejorar los procesos definidos para el tratamiento de la obesidad.

El principal objetivo de este proyecto de investigación es la especificación de un modelo de representación visual 4D (3D + Tiempo) del cuerpo humano para llevar a cabo un análisis de la evolución morfológica ocasionada por cambios producidos por el tratamiento de la obesidad (ver Figura 1.1).

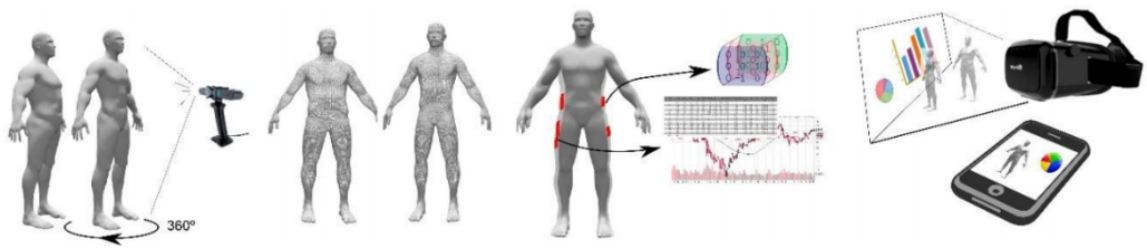

Figura 1.1: Pipeline para el proyecto de investigación Tech4diet.

Se pretende generar un modelo 3D que se utilizará para el desarrollo de un sistema de visualización del cuerpo que mejore el tratamiento de la obesidad. Una vez desarrollado el módulo que genere este modelo se pretende proporcionar visualizaciones mediante realidad virtual inmersiva y simulaciones de la evolución consecuencia del tratamiento. Este proceso se lleva a cabo con la intención de poder estudiar la repercusión de estas visualizaciones realistas en la adherencia del paciente al tratamiento a lo largo de su desarrollo.

Además, se pretende proporcionar herramientas fundamentadas en el modelo que faciliten el estudio de nuevos índices basados en medidas antropométricas y su evolución en el tiempo.

La generación de modelos virtuales en tres dimensiones que representen la forma y apariencia de objetos reales es un proceso que cada día cuenta con más utilidades en una amplia variedad de sectores. Para llevar a cabo esta tarea contamos con tantas posibilidades casi como utilidades y además estas crecen conforme vamos creando hardware más potente. La investigación en este sector siempre se ha caracterizado por sus retos poco triviales y es que la reconstrucción de objetos tridimensionales a partir de cuerpos reales está considerada como un complejo problema científico.

La demanda de llevar a cabo estas tareas de forma eficaz y precisa se está viendo fuertemente impulsada por el mercado emergente que surge hace escasos años enfocado en ámbitos como la realidad virtual, la realidad aumentada y la realidad mixta. La

incorporación de estas tecnologías en multitud de procesos de nuestra vida cotidiana está requiriendo que el escaneado de objetos y la integración de estos en un mundo virtual se realice en tiempo real y con unos estándares de calidad bastante elevados.

En el contexto del proyecto de investigación el presente trabajo aborda la adquisición 3D del cuerpo humano a partir de redes de cámaras RGB-D, la obtención del modelo 3D del cuerpo, su representación texturizada y la exportación del modelo para su posterior uso mediante realidad virtual. En la propuesta planteada se ha dividido el proceso en siete fases (ver Figura 1.2):

a. **Fase de calibrado**: en la que se ajustarán los distintos dispositivos tanto de forma intrínseca como extrínseca.

b. **Fase de adquisición**: en la que se captarán las distintas imágenes y nubes de puntos.

c. **Fase de preprocesado**: en la cual cada una de las nubes de puntos serán optimizadas para las siguientes fases.

d. **Fase de registro**: en la que se aplicarán las transformaciones a las nubes de puntos y se generará una nube registrada.

e. **Fase de modelado**: en la que se generará una malla a partir de la nube de puntos.

f. **Fase de texturizado**: Se proyectarán sobre la malla las distintas vistas aportando así color a sus polígonos y se guardará esta asignación en forma de textura.

g. **Fase de exportación**: Como último se generará un modelo exportable que se pueda utilizar en la etapa de visualización.

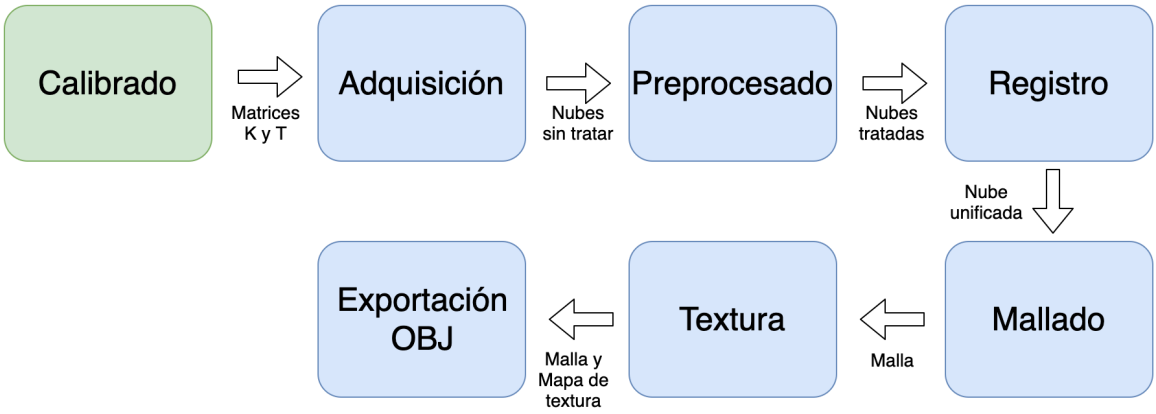

Figura 1.2: Pipeline del proceso de adquisición.

1.2 Estado del arte

Una vez establecido el contexto del proyecto en el ámbito de la adquisición 3D del cuerpo humano a partir de redes de cámaras RGB-D, la obtención del modelo 3D del cuerpo y su representación texturizada, se realiza el siguiente estudio de los diferentes métodos utilizados para resolver los problemas que se plantean. El estado del arte se ha organizado de la siguiente forma:

En primer lugar se hace una reflexión sobre el enfoque global del problema y a continuación se estudia la visión estereoscópica y su funcionamiento, se continúa valorando las opciones a la hora de efectuar un calibrado en un sistema de visión 3D, posteriormente se tratan las opciones que se emplean para preprocesar las nubes de puntos y para finalizar, como generar una malla a partir de estas y texturizarla.

Este proyecto está fuertemente enfocado al modelado de cuerpos, y contamos con todo un mundo de posibilidades y opciones dentro de este ámbito. En base a un primer estudio realizado sobre la aproximación del problema y como se estaba afrontando actualmente, vi que dos de las opciones que más estaban empleándose eran, por un lado realizar el escaneado corporal y generación del registro (alineamiento y unificación

de distintas nubes de puntos) con una única cámara RGB-D desde la que obtenemos una nube de puntos ubicada en el espacio 3D, posteriormente extraeríamos los puntos característicos de la misma y la solaparemos con la siguiente adquisición mediante el emparejamiento de sus respectivos puntos, generando de esta forma el registro.

Por otro lado, se propone la obtención de las nubes de puntos mediante varias cámaras RGB-D generando el registro completo mediante la transformación y unión de estas en función de los parámetros obtenidos en el calibrado previo. En concreto, después de reflexionar sobre los puntos fuertes y débiles de cada opción y ver que se pueden obtener resultados de una elevada calidad con ambos sistemas, este último es el método que finalmente he decidido utilizar por tener las características de ser un método más técnico y con una componente de gestión de hardware más elevada, tema que se alinea de una forma más adecuada con mis habilidades e intereses.

1.2.1 Visión estereoscópica

Cuando se habla de visión estereoscópica o visión estéreo nos referimos a un sistema de captación formado habitualmente por dos cámaras, usualmente izquierda y derecha. Las características que aparecen en las imágenes captadas por las cámaras aparecerán en distinta posición en una imagen y en la otra. Cuanto más cerca estén los objetos de las cámaras, mayor será su cambio relativo entre la posición del objeto en la imagen de una y otra cámara. El desplazamiento relativo de los objetos en las dos imágenes se denomina disparidad y existe una relación directa entre la profundidad y la disparidad.

Figura 1.3: Imágenes izquierda y derecha.

Un mapa de disparidad está generado a partir de dos imágenes, para este caso izquierda y derecha (ver Figura 1.3), y refleja la diferencia de posición de los puntos característicos de una imagen respecto a la otra. El mapa de disparidad tiene el aspecto de la siguiente imagen (ver Figura 1.5), y desde él, mediante la ecuación mencionada a continuación (ver Figura 1.4) en la que se expresa la relación entre disparidad (d) y profundidad (z) empleando la focal (f) en pixels y el Baseline (B), este último en la unidad de medida que queramos obtener el resultado, usualmente en m o mm, y gracias a la cual podemos extraer un mapa de profundidad (ver Figura 1.6) y por consecuencia, una nube de puntos de la escena.

$$z = \frac{f \cdot B}{d}$$

Figura 1.4: Relación para obtener profundidad.

Figura 1.5: Mapa de disparidad.

Figura 1.6: Mapa de profundidad.

Una nube de puntos es un conjunto de puntos formados por las coordenadas de su posición con respecto al origen del sistema. Cada uno de estos puntos se identifica habitualmente como una 3-tupla de coordenadas cX, cY, cZ.

Aparte de esta información que corresponde a la ubicación del punto en el espacio, por cada punto se pueden incluir diversos atributos pero principalmente se ha utilizado el color, en formato RGB, representado mediante una 3-tupla con un elemento para cada uno de sus componentes. La normal, representada por otra 3-tupla nX, nY, nZ que indica la dirección a la que estaría orientada la superficie local formada por el punto actual con los puntos vecinos.

Como veremos en el capítulo 4.2, el hardware utilizado para obtener las imágenes emplea visión estereoscópica activa. La visión estereoscópica activa es un tipo de visión estereoscópica que proyecta sobre la escena una luz como un láser o una luz estructurada para optimizar el cálculo de la profundidad en base a la interacción del patrón con la escena a captar.

Los sistemas de profundidad estereoscópicos clásicos, que como ya hemos visto se basan en características de la imagen para extraer el mapa de disparidad, encuentran bastantes problemas al intentar extraer la profundidad en superficies sin excesivos puntos de referencia o *keypoints*, como pueden ser superficies lisas con ligeras curvas

y sin textura. Una forma de solventar este problema es proyectar un patrón sobre la escena, y este servirá como textura y por lo tanto como *keypoints* de la escena, aunque originalmente no las tuviese.

En concreto el hardware empleado (Intel RealSense D435) proyecta un patrón infrarrojo aleatorio (ver Figura 1.7) por cada dispositivo y cada par de cámaras en un mismo dispositivo utiliza el mismo patrón. Este motivo hace que no sea posible la interferencia entre patrones cuando tengamos presente la superposición de varios en la misma escena.

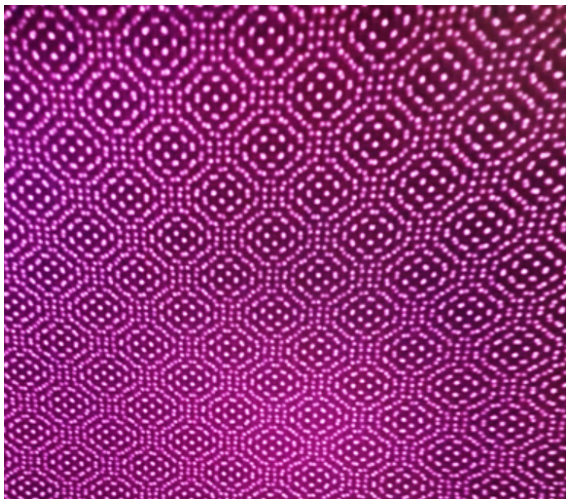

Figura 1.7: Patrón proyectado sobre una pared plana.

1.2.2 Calibrado de sistemas de visión 3D

Cuando llevamos a cabo un calibrado estamos realizando un proceso que nos permitirá obtener los parámetros que, aplicados a la información que percibimos a través del sensor o los sensores, tratará de que esta o estas, se asemejen lo mejor posible a la realidad tal y como la percibimos. Cuando calibramos un sistema tenemos dos tipos de parámetros que pueden definir su calibración, los intrínsecos y los extrínsecos.

1.2.2.1 Parámetros intrínsecos

Estos parámetros definen la geometría interna de la cámara y son parámetros que, aunque suelen ser suministrados por el fabricante de la cámara, suelen variar ligeramente debido a las imperfecciones del proceso de fabricación. Debido a esto, no hay dos sensores exactamente iguales y aplicar unos valores de corrección estándar, hace que el sensor genere una visión distorsionada de la realidad. Durante este proceso tratamos de aproximar, de la manera más precisa posible, los parámetros que definen este sensor evitando así desmontarlo y medirlo, con la dificultad que esto conlleva.

Con el calibrado de sensores RGB-D tenemos dos tipos de sensores que definir, el sensor o los sensores que perciben la información de color (ver Figura 1.8) y los sensores que nos aportan la información de profundidad (ver Figura 1.9).

Figura 1.8: Información de color.

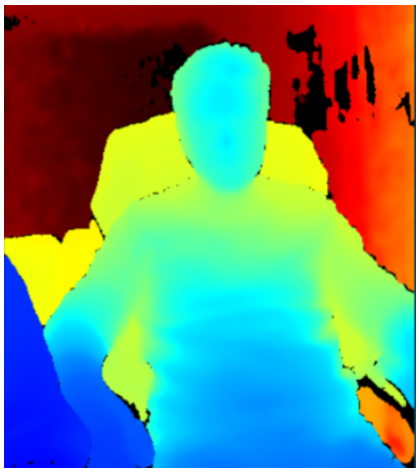

Figura 1.9: Información de profundidad.

Hay multitud de métodos para llevar a cabo la extracción de los parámetros intrínsecos de un dispositivo (Villena Martínez, 2015) pero en este caso he empleado el proporcionado por la firma Intel tal y como se detallará en el capítulo 3.1 en el que se aborda el desarrollo del calibrado extrínseco.

Para la calibración intrínseca contamos con los siguientes parámetros que averiguar durante la calibración:

- Distancia focal. Es una propiedad física que poseen las lentes. Esta propiedad determina la distancia a la que se encuentra el punto a través del cual pasan todos los rayos paralelos al eje óptico que atraviesan la lente (ver Figura 1.10). Por otro lado, en fotografía, la distancia focal es la distancia que existe entre el centro óptico y el sensor fotosensible el cual está situado en el plano focal donde se formará la imagen (ver Figura 1.11).

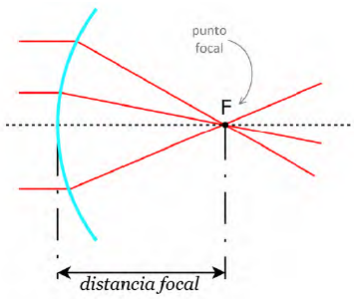

Figura 1.10: Distancia focal de una lente.

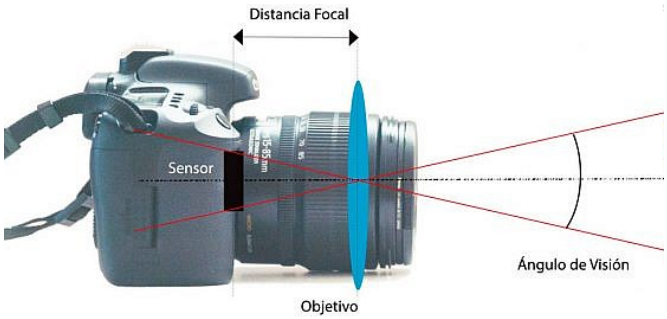

Figura 1.11: Distancia focal en fotografía.

En la mayoría de los casos, las células de los sensores no son cuadradas y son rectangulares, por esta razón manejamos dos magnitudes para la distancia focal, una horizontal y otra vertical, f_x y f_y.

- Field of View, o campo de visión, es la distancia cubierta por la cámara a una cierta profundidad. Por otro lado, se define el ángulo de visión como la porción de la escena incluida en la imagen, es decir, el número de grados incluidos en la imagen. Por ejemplo, una cámara con una ángulo de visión de 90º a un metro de distancia tendrá un campo de visión de 2 metros (ver Figura 1.12)

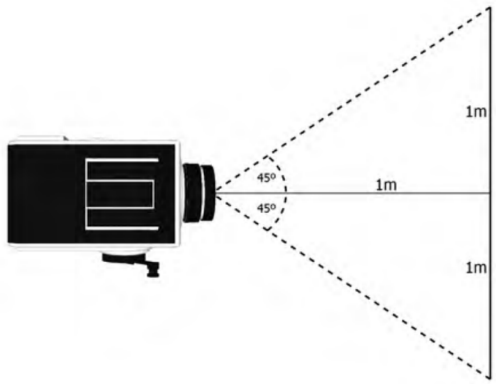

Figura 1.12: Ejemplo FOV de una cámara.

- El punto principal. Representa el desplazamiento del eje óptico respecto al sensor, lo que provoca un desplazamiento del centro de proyección en la imagen.

  Tal y como se ve en la imagen de la izquierda (ver Figura 1.13), en una imagen de dimensiones w x h el punto principal perfecto debería de encontrarse en el punto (w/2 = c_x, h/2 = c_y) sin embargo, debido al desplazamiento del eje óptico lo encontramos en el punto (u,v).

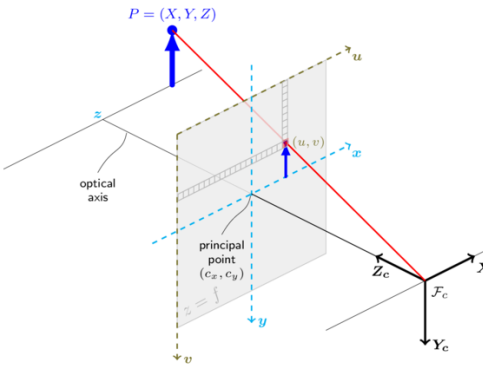

Figura 1.13: Desplazamiento del punto principal.

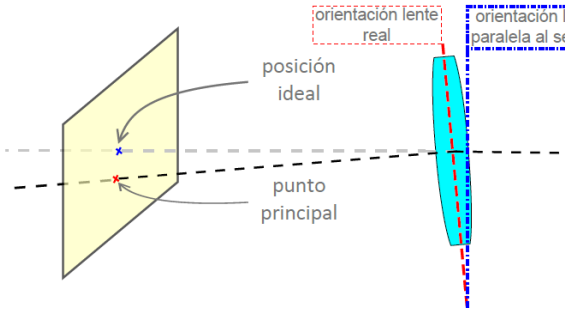

Figura 1.14: Motivo desplazamiento punto principal.

En la imagen de la derecha (ver Figura 1.14) podemos observar el conjunto azul, que definiría un alineamiento perfecto del eje óptico y del punto principal con el centroide del sensor. Por otro lado, en el conjunto rojo se puede observar que debido

al fallo en la orientación de la lente, se produce una distorsión que provoca un desplazamiento en la posición del punto principal respecto al centroide del sensor.

Formalmente los valores para distancia focal y *punto principal* vienen representados en una matriz K de la siguiente forma (ver Figura 1.15).

$$\mathrm{K} = \begin{bmatrix} \frac{f}{p_x} & 0 & c_x \\ 0 & \frac{f}{p_y} & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Figura 1.15: Representación de valores en matriz K.

- Distorsiones ópticas

Otro efecto producido por defectos o fallos en el diseño de las lentes y que también podemos tener en cuenta en el calibrado son las distorsiones ópticas. Los dos tipos más habituales de distorsión son la tangencial y la radial.

La **distorsión tangencial** está ocasionada por el descentrado de la lente, o lo que es lo mismo, que la lente no esté ubicada de forma totalmente paralela al sensor. Este desplazamiento ocasiona distorsiones en la imagen, entre ellas el desplazamiento del punto central en la imagen (Ghosh, 2005). Como se puede observar en el siguiente gráfico (ver Figura 1.16) la distorsión radial produce un desplazamiento en el radio dr tomando como centro el punto principal. El desplazamiento de la segunda distorsión está producido de forma angular con una cantidad dt, pivotando sobre el punto principal.

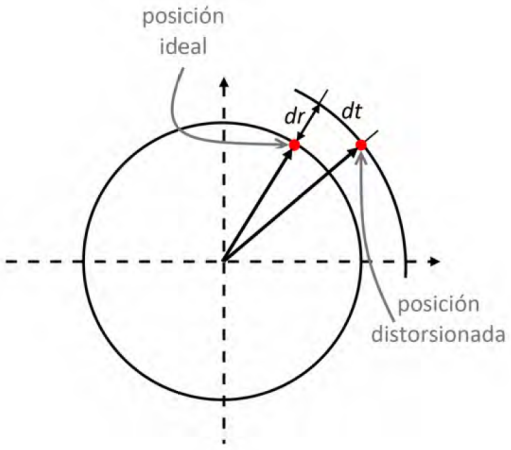

Figura 1.16: Efecto de las distorsiones radial y tangencial.

En las cámaras de bajo coste las lentes suelen ser esféricas en lugar de ser parabólicas ya que esto supone una reducción de costes a la hora de su elaboración y este es uno de los motivos que ocasionan la distorsión radial.

Como podemos ver en la imagen (ver Figura 1.17), esto produce que los haces de luz que atraviesan la lente esférica no converjan en el mismo punto que los haces más cercanos al centro. Este efecto provoca que la imagen no se forme de una forma correcta en el plano correspondiente. No ocurre lo mismo cuando la lente está elaborada de forma parabólica, tal y como se puede observar en el ejemplo de la derecha, en la que todos los haces convergen en el punto focal.

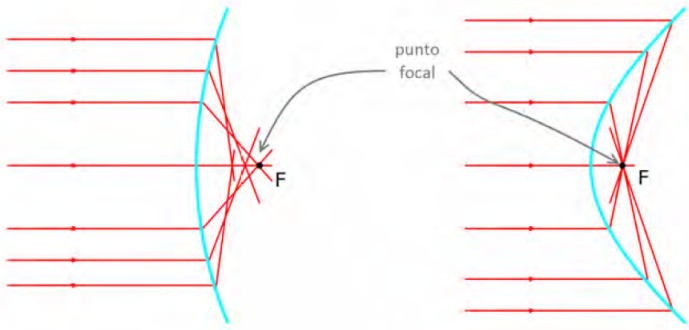

Figura 1.17: Distorsión radial en lentes esféricas (izq.) y parabólicas (dcha.).

La **distorsión radial** produce diferentes efectos de distorsión en la imagen, unos de los más destacados son las distorsiones: de barril, cojín y bigote.

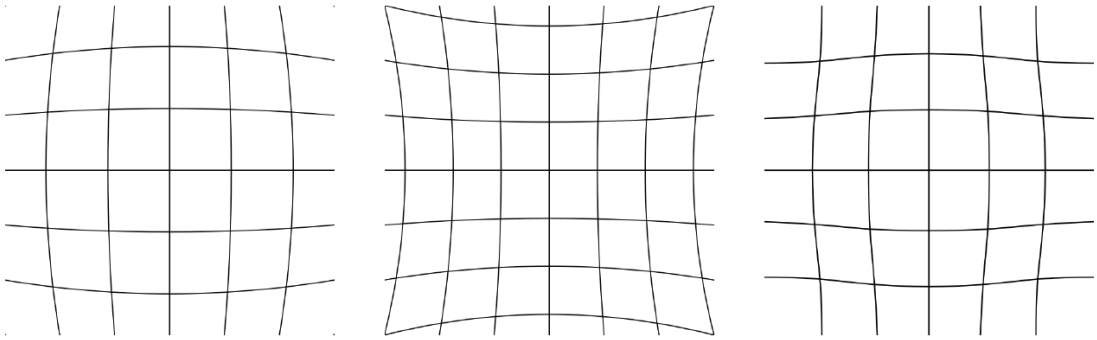

Figura 1.18: Distorsiones radiales ópticas.

- La distorsión de barril, *barrel o fish eye distortion* (ver Figura 1.18a), ocasiona que, según nos alejemos del punto principal, los píxeles se alejen de este de forma proporcional, es el efecto más común en los sensores RGB-D.
- La distorsión de cojín o *pincushion distortion* (ver Figura 1.18b), produce el efecto contrario que la distorsión de barril, desplazando los píxeles hacia el centro de la imagen conforme nos alejamos del punto principal.
- La distorsión de bigote o *moustache distortion* (ver Figura 1.18c), produce un efecto combinado entre los dos anteriores, provocando una distorsión de barril que se convierte de forma progresiva en distorsión de cojín conforme aumenta la longitud del radio con el punto principal.

1.2.2.2 Parámetros extrínsecos

Los parámetros extrínsecos definen la posición del sensor en un sistema de coordenadas global. Junto con los parámetros intrínsecos nos permiten averiguar la posición 3D global de cierto punto en el sistema de coordenadas 2D de la imagen.

De forma contraria a los intrínsecos, estos valores no están ligados a un sensor de forma fija, si no a un conjunto de estos y varían en función de la pose de cada uno de ellos respecto al sistema de coordenadas. En los dispositivos con varios sensores se suele situar la referencia en uno de ellos.

La expresión de estos valores se realiza mediante dos matrices. Por un lado tenemos la matriz de 3x3 R de rotación, que se compone de la rotación acumulada en cada uno de los ejes para cuadrar el sistema de coordenadas global con el de la cámara. Por otro lado tenemos la matriz de 1 x 3 t de traslación que nos indica el desplazamiento en los ejes x, y, z para cuadrar los orígenes de los sistemas de coordenadas. Estas dos matrices se unen en una matriz de 3 x 4 (ver Figura 1.19). Habitualmente se añade una fila inferior denominada coordenada homogénea con los valores 0001, para convertir la matriz a cuadrada con dimensiones 4 x 4 y facilitar la operatividad.

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}$$

Figura 1.19: Matriz de calibración extrínseca.

1.2.2.2.1 Alineamiento de sensores

Como ya he comentado en el calibrado, los sistemas formados por más de un sensor generan escenarios en los que nos encontramos con varias imágenes con sistemas de referencias distintos. Para unificar el eje de coordenadas entre ellos aplicamos las transformaciones necesarias las cuales habremos obtenido a través del calibrado extrínseco del conjunto. La distancia que separan dos sensores en un sistema se conoce como *Baseline*, esta distancia hay que obtenerla mediante el calibrado (ver Figura 1.20) y lo mismo sucede con la orientación ya que no es común que los sensores se encuentren ubicados de forma completamente paralela.

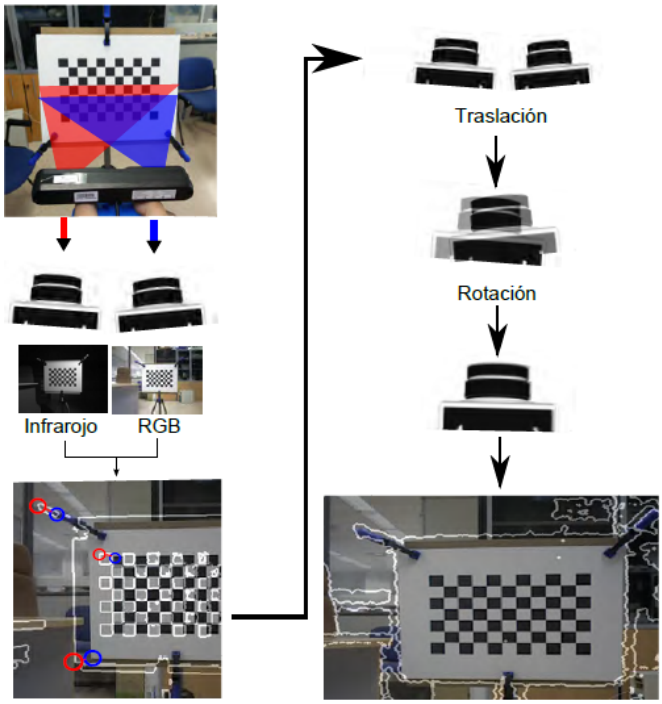

Figura 1.20: Alineamiento entre imágenes RGB y profundidad.

Se conoce como alineamiento al proceso de situar dos imágenes obtenidas por diferentes sensores en un mismo sistema de referencia. Para realizar este proceso previamente debemos conocer las matrices de rotación y traslación. En la siguiente imagen (ver Figura 1.21) tenemos dos conjuntos de puntos con sistemas de referencia distintos. Tenemos el objetivo de alinear el conjunto de la izquierda con el de la derecha, para lo que necesitaremos aplicar una rotación que haga coincidir la orientación de los puntos y una traslación que los desplace para hacerlos coincidir en posición.

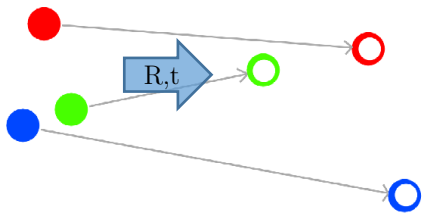

Figura 1.21: Transformación para alinear dos conjuntos.

Aplicaremos estas transformaciones a la nube de puntos para alinearlas con la de color y ya podremos proyectarla sobre la imagen para averiguar las correspondencias entre los puntos y los pixeles.

1.2.2.2.2 Calibración extrínseca entre dispositivos

Cuando hablamos de calibración extrínseca entre dispositivos, a diferencia de la calibración extrínseca entre sensores de un mismo dispositivo o alineamiento de sensores, nos referimos a la ubicación de los distintos dispositivos o cámaras del sistema en un mismo mundo, en el que comparten el sistema de coordenadas, tal y como se puede ver en la siguiente imagen (ver Figura 1.22).

Mediante este calibrado obtenemos una matriz T para cada una de las cámaras, la cual, aplicada al conjunto de datos entregado por este dispositivo, nos transforma este conjunto de forma que comparte el origen de coordenadas con el resto.

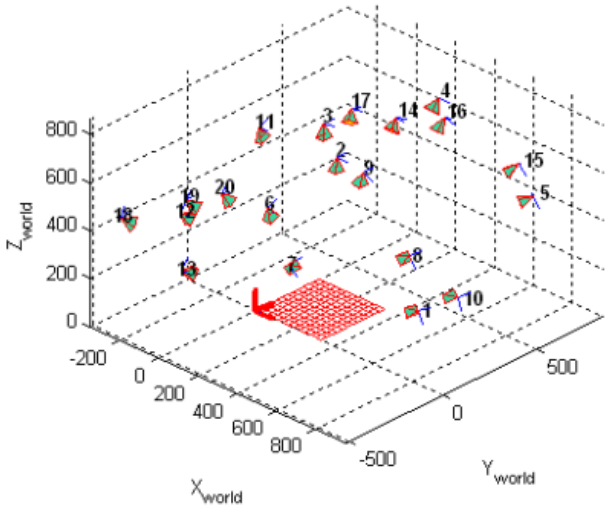

Figura 1.22: Red de cámaras en un mismo espacio 3D.

1.2.3 Preprocesado y filtrado de las nubes de puntos

La optimización de las nubes de puntos antes de llevar a cabo el registro es un paso clave en el proceso a la hora de conseguir un resultado óptimo. Por este motivo el filtrado es un área en constante investigación. Los principales enfoques de filtrado

para la nube de puntos 3D pueden clasificarse en cuatro grupos: filtrado basado en estadísticas, basado en vecindad, basado en proyecciones y basado en PDEs (derivadas parciales) (Han et al., 2017).

En concreto para este trabajo se han empleado filtros pertenecientes a los dos primeros grupos como son el filtro de mediana, el filtro bilateral y el Statistical Outlier Removal, los cuales han sido tratados con profundidad en la sección de preprocesado.

1.2.4 Generación de malla desde nubes de puntos

El proceso de mallado tiene como objetivo conseguir una reconstrucción de una superficie, generalmente formada por polígonos, a partir de una nube de puntos. A este resultado, que no es más que un conjunto de vértices relacionados y que forman caras con una orientación concreta, le denominamos malla.

Actualmente este problema se afronta desde múltiples enfoques según el tipo de superficie que tengamos como objetivo mallar (Salman, Yvinec, Mérigot, & Merigot, 2010), en concreto me he centrado en tres enfoques de los más populares que combinan la reconstrucción y la generación de malla como son el algoritmo Marching cubes que es capaz de detectar y clasificar de forma fiable las regiones de características nítidas en la superficie y muestrearlas con precisión (Kobbelt, Botsch, Schwanecke, & Seidel, n.d.). El algoritmo Greedy Projection Triangulation que se recoge dentro del grupo de algoritmos iterativos de punto más cercano y combina esta técnica con la localización de descriptores FPFH (Liu, Bai, & Chen, n.d.). Y finalmente el algoritmo Poisson que busca la función que mejor concuerda con el conjunto de observaciones ruidosas y no uniformes, y que tiene la capacidad de recuperar robustamente los detalles finos de los escaneos ruidosos del mundo real. Se pueden ver distintas pruebas con cada uno de ellos en el capítulo 4.3.

1.2.5 Proyección de textura sobre malla

La proyección de una textura sobre una malla es un proceso habitual y trivial en la generación de gráficos 3d para videojuegos, pero cuando se trata de proyectar imágenes obtenidas del mundo real mediante una o varias cámaras en un modelo virtual, el problema aumenta su grado de complejidad.

En el proceso de construcción de un modelo virtual 3D realista a partir de escenas reales, a menudo se necesitan múltiples mapas de texturas de color. Cuando tomamos imágenes desde diferentes ángulos de visión y con diferentes configuraciones de cámara se crean colores que no coinciden debido a la iluminación y a las condiciones de la cámara. Como algunos investigadores han demostrado, las texturas de dos imágenes diferentes causan discrepancias de color y suele concluir con una incómoda apariencia de mosaico en las superficies (Bannai, Fisher, & Agathos, 2006).

Hay diversos estudios y papers sobre la materia, tras varias pruebas, en este trabajo se ha optado por utilizar el método *Masked photo blending: Mapping dense photographic data set on high-resolution sampled 3D models* implementado en la API de Meshlab que evalúa la calidad local y geométrica de la textura por cada uno de los píxeles mediante el uso de funciones de ponderación simples reduciendo artefactos de iluminación e incoherencias entre las diferentes imágenes (Callieri, Cignoni, Corsini, & Scopigno, 2008). El funcionamiento del algoritmo se detalla en el capítulo 4.4.

1.3 Objetivos

El objetivo general del proyecto de investigación en el que se enmarca este trabajo es la obtención de métodos computacionales para estudiar la evolución de la apariencia visual del cuerpo humano usando técnicas de visión 3D/4D para mejorar el diagnóstico, el tratamiento de la obesidad y la adherencia a los tratamientos dietéticos.

En este marco, el presente trabajo tiene como objetivo global proporcionar métodos para la adquisición 3D del cuerpo humano, su modelado 3D, así como su visualización texturizada. Para la consecución de este objetivo global se plantean los siguientes objetivos parciales:

1.3.1 Objetivo 1: Calibrado

Calibrado de la red de cámaras RGB-D y adquisición de las múltiples vistas 3D del cuerpo humano. Tareas:

a. Diseño del entorno de captura. Dimensionamiento de la estructura de la red de cámaras RGB-D y selección de estas.

b. Diseño del entorno de procesamiento y comunicaciones. Seleccionar del hardware y software para gestionar la red de cámaras y su sincronización.

c. Montaje de hardware e instalación de software del sistema.

d. Selección de los métodos de calibrado intrínseco y extrínseco.

e. Implementación de la adquisición.

1.3.2 Objetivo 2 Preprocesamiento

Filtrado de las nubes de puntos adquiridas mediante el sistema de adquisición 3D multivista. Tareas:

a. Selección de métodos de filtrado para tratar las nubes de puntos obtenidas.

b. Implementación y test de las opciones de optimización y filtrado de las nubes de puntos.

1.3.3 Objetivo 3 Registro

Registro de las diversas nubes de puntos en una única nube, compartiendo un espacio de coordenadas común. Tareas:

a. Selección de métodos de registro.

b. Implementación y test de los métodos de registro.

1.3.4 Objetivo 4 Mallado

Obtención del modelo 3D basado en malla. Tareas:

a. Selección de métodos de mallado

b. Implementación de los métodos de obtención de la malla.

1.3.5 Objetivo 5 Texturizado

Obtención del mapa de texturas para su visualización realista. Tareas:

a. Selección de métodos de proyección de imágenes sobre malla.

b. Implementación de los métodos de texturizado.

c. Exportación del modelo de malla texturizado.

1.4 Planificación

El inicio del desarrollo de este trabajo comenzó en febrero de 2018 y la planificación del mismo en el tiempo ha sido determinada por la planificación desarrollada para el proyecto de investigación al que pertenece. Pese a que la distribución de este está fijada a un plazo de 3 años, la parte que ha motivado este trabajo cubre una primera etapa del proceso que finaliza en mayo de 2019. En el planteamiento del proyecto de investigación se puede observar el *deadline* donde se requiere tener el sistema de adquisición listo para empezar a generar el data-set de modelos 3D para el estudio, este punto se puede observar en el siguiente diagrama de Gantt (ver Figura 1.23).

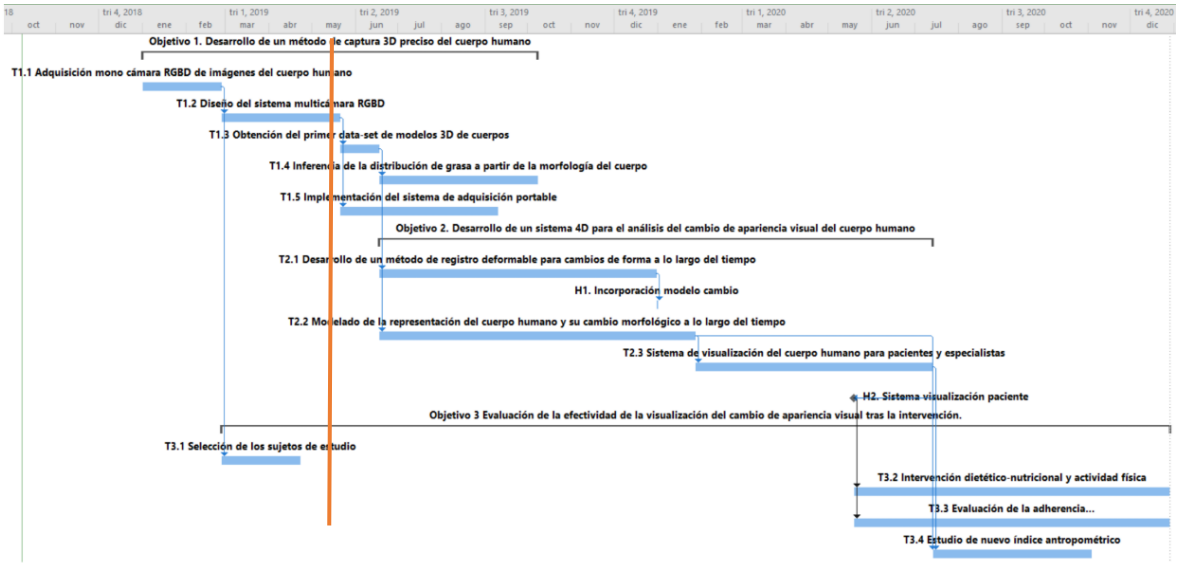

Figura 1.23: Gantt de planificación del proyecto de investigación Tech4Diet.

1.4.1 Tecnologías de organización

En este capítulo se hace una mención a las dos herramientas empleadas para coordinar la organización con el equipo, que han sido Bitbucket y Trello.

1.4.1.1 BitBucket

Figura 1.24: Logotipo Bitbucket.

Bitbucket es un servicio de alojamiento basado en la web para los proyectos que utilizan el sistema de control de versiones Mercurial y Git, que nos permite administrar repositorios de forma privada en su versión gratuita. Se ha empleado esta herramienta para llevar un control de versiones entre los distintos componentes del equipo.

1.4.1.2 Trello

Figura 1.25: Logotipo Trello.

Trello es un software de administración de proyectos con interfaz web y con cliente para iOS y Android que permite organizar el trabajo mediante asignación de tareas entre los componentes del equipo.

1.4.2 Forma de organización

El desarrollo de este trabajo, desde su inicio en febrero de 2018 hasta su finalización en mayo de 2019 ha abarcado un periodo de 15 meses. A pesar de que el trabajo no tenía unas dependencias muy marcadas en el tiempo entre los miembros del equipo, ha funcionado de forma sincronizada durante todo el proceso. Con esto hemos conseguido recibir el apoyo entre los miembros en los momentos en los que se producían atascos o cualquier tipo de bloqueo. En el grupo de investigación hay miembros con una dilatada experiencia y recursos propios y, gracias a la metodología y el sistema de trabajo empleados, hemos sido capaces de sacar el máximo partido a estas cualidades en los momentos en los que los componentes con menos experiencia hemos necesitado apoyo para solucionar dudas. Para organizarnos de una forma eficiente hemos empleado distintas herramientas las cuales han sido claves para el correcto desarrollo y la fluidez del trabajo, como son Trello, Whatsapp, Skype y Bitbucket.

1.4.3 Metodología

Aunque he trabajado de forma autónoma en el desarrollo del sistema, en todo momento he ido avanzando de forma coordinada y compartiendo mis hitos, problemas y logros con el resto del equipo. Como sistema de trabajo hemos empleado el desarrollo bajo las metodologías ágiles, en concreto hemos utilizado la metodología SCRUM.

El desarrollo de esta metodología se realiza de forma iterativa e incremental. Cada iteración se denomina Sprint y tiene una duración preestablecida de entre 1 y 4 semanas. Al finalizarla se obtiene como resultado una versión del software con nuevas

prestaciones listas para ser usadas. En cada nueva iteración o Sprint se va ajustando la funcionalidad ya construida y se añaden nuevas prestaciones, dando prioridad a aquellas que aporten mayor valor.(Schwaber & Sutherland, 2016)

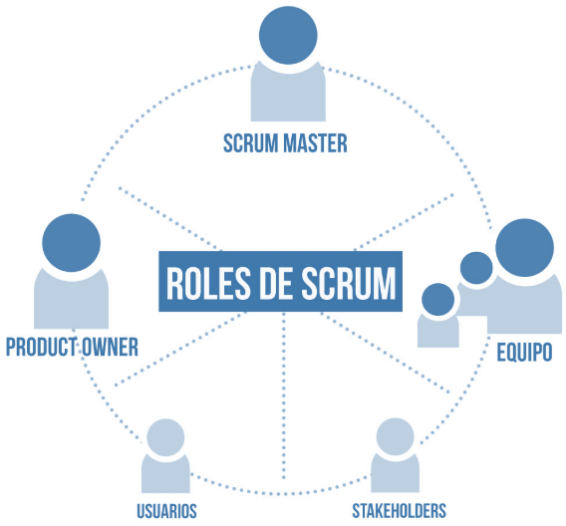

Figura 1.26: Roles de Scrum.

Como vemos en la imagen anterior (ver Figura 1.26) podemos encontrar hasta cinco, pero los roles más importantes dentro de esta metodología son tres:

- **Product owner:** Es la persona que va a usar el software o su representante. Genera las historias en base a las necesidades y les asigna la prioridad de forma constante.
- **Scrum master:** Es el encargado de guiar al equipo para cumplir de forma correcta la metodología y reducir la gestión de los impedimentos. A su vez trabaja con el Product Owner para mejorar el retorno de la inversión.
- **Equipo:** Conjunto de profesionales técnicos con las habilidades necesarias para desarrollar las historias en cada uno de los sprints.

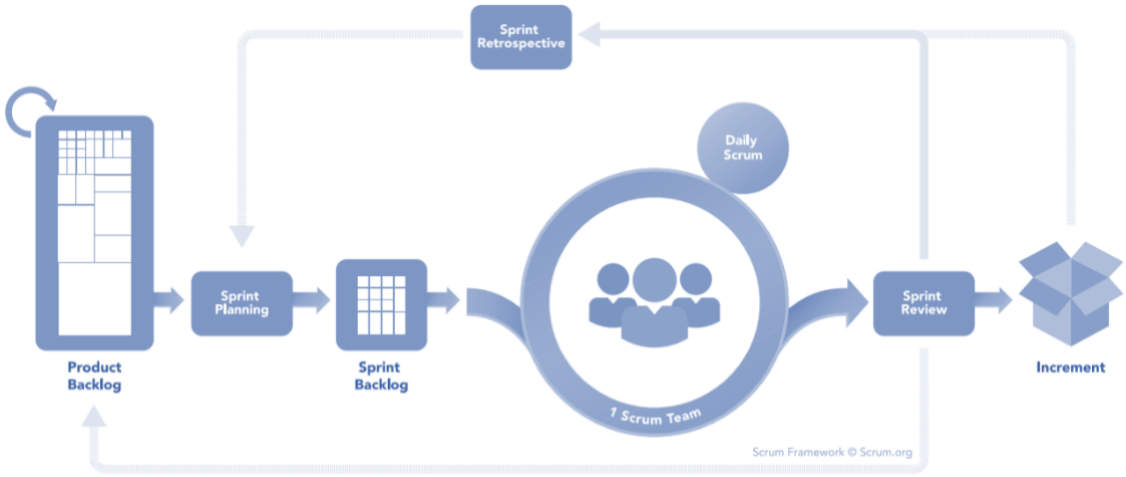

Figura 1.27: Framework de la metodología Scrum.

Como se puede observar en el esquema (ver Figura 1.27) esta metodología consta de seis fases bien definidas:

- **Product Backlog**: Es el conjunto de requisitos, también denominados historias expresados en un lenguaje no técnico y jerarquizados en función del valor que aporten. Los requisitos y prioridades no son fijos y se revisan y ajustan a lo largo del proyecto de forma regular.
- **Sprint Planning:** Reunión mediante la cual el Product Owner expone las historias del backlog en función de su prioridad. Los miembros del equipo se seleccionan la cantidad de historias a desarrollar en el sprint, para a continuación planificar su desarrollo durante este.
- **Sprint:** Iteración de duración previamente establecida en la cual el equipo implementa las historias determinadas en el sprint planning para finalizar con una versión del software totalmente operativa.
- **Sprint Backlog:** Listado de tareas a llevar a cabo para completar el conjunto de historias seleccionadas para el sprint.

- **Daily scrum:** Reunión diaria de corta duración en la que el equipo comenta, que hizo ayer, que hará hoy y si ha tenido algún problema durante el desarrollo.
- **Sprint review:** Reunión en la que el equipo presenta las historias conseguidas y se comenta que se hizo bien, que es mejorable y se define como mejorarlo.

Es cierto que al no ser un equipo muy grande de trabajo hemos aligerado un poco su estructura prescindiendo de algunos de sus roles, o asignando varios a un mismo componente del equipo.

Los sprints han sido de carácter semanal y teníamos establecido un día a la semana para revisar las tareas llevadas a cabo en el sprint, cerrar el incremento y planificar el siguiente. De forma diaria hemos puesto en común el trabajo realizado con los compañeros directos y cada mes o mes y medio aproximadamente, hemos tenido reuniones con los equipos de nutrición y psicología para ir recibiendo *feedback* de que el producto desarrollado está cumpliendo las expectativas de uso necesarias.

1.4.4 Diseño plan de trabajo

Mi parte del trabajo, aunque he ido compartiendo los avances con el resto del equipo semanalmente para ir trabajando en acorde, ha estado organizada de forma individual.

Debido a la necesidad de cumplir los plazos establecidos por el proyecto de investigación decidí organizar las tareas a llevar a cabo mediante *Scheuduling* valiéndome de la herramienta Microsoft Project. El resultado se puede ver en las siguientes imágenes (ver Figura 1.28 y Figura 1.29). Como se aprecia, el trabajo se inicia en febrero de 2018 y concluye el 24 de mayo de 2019, prácticamente un mes antes del *deadline* establecido para la continuación del proyecto. He decidido dejar

esta holgura por si hubiese cualquier tipo de problema no tener que retrasar el trabajo en el proyecto de investigación.

Nombre de tarea	Duración	Comienzo	Fin	Prec
Estudio estado del arte	15 días	jue 15/02/18	mié 07/03/18	
Elección del entorno de trabajo	15 días	jue 08/03/18	mié 28/03/18	1
Elección hardware servidor	15 días	jue 29/03/18	mié 18/04/18	2
Elección herramienta para tratar nubes de puntos	10 días	jue 19/04/18	mié 02/05/18	3
Estudio sistemas de calibrado	20 días	jue 03/05/18	mié 30/05/18	4
Estudio estructura de adquisición	5 días	jue 31/05/18	mié 06/06/18	5
Montaje e instalación de servidor	10 días	jue 07/06/18	mié 20/06/18	6
Implementación del proceso de gestión de las cámaras	45 días	jue 21/06/18	mié 22/08/18	7
Implementación del proceso de gestión de las nubes de puntos	45 días	jue 23/08/18	mié 24/10/18	8
Implementación del proceso de gestión de mallado	30 días	jue 25/10/18	mié 05/12/18	9
Implementación del proceso de gestión del texturizado	30 días	jue 06/12/18	mié 16/01/19	10
Integración del conjunto	45 días	jue 17/01/19	mié 20/03/19	11
Testeo y calibrado de parámetros de filtros	25 días	jue 21/03/19	mié 24/04/19	12

Figura 1.28: Tareas a llevar a cabo en este trabajo.

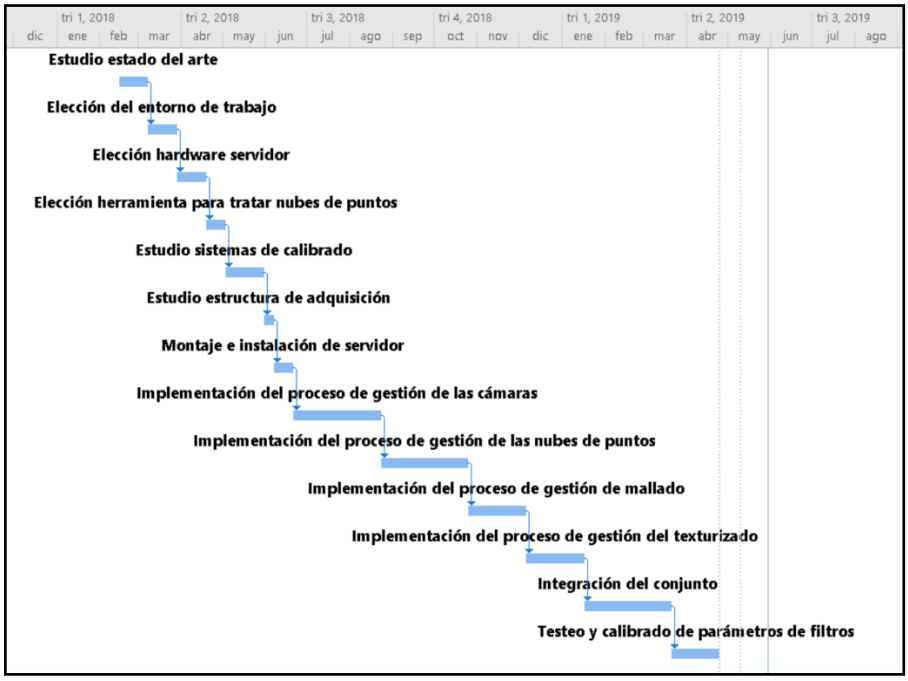

Figura 1.29: Consecución de las tareas definidas en la figura 1.26.

2 ENTORNO Y TECNOLOGÍAS DE DESARROLLO

2.1 Arquitectura del sistema

En este apartado se van a desarrollar las decisiones tomadas para llegar a tener el servidor con el que se trabaja hoy en día en el proyecto, tanto su formato físico como su software.

2.1.1 Estructura de adquisición

Para la estructura de adquisición inicialmente y en base al *Field of View* de las cámaras se optó por diseñar una estructura de perfil de aluminio de 80 mm con unas dimensiones fijas de 2000 x 2200 mm como se puede ver en la figura de la izquierda (ver Figura 2.1). Finalmente y con objetivo de poder ajustar las medidas durante la investigación, se optó por hacer 4 postes del mismo perfil de aluminio, con la misma altura, pero con una base que me permitiese moverlos, con el fin de, una vez estén testeadas las medidas, hacer la estructura definitiva en el formato fijo. Este diseño se puede ver en la imagen de la derecha (ver Figura 2.2).

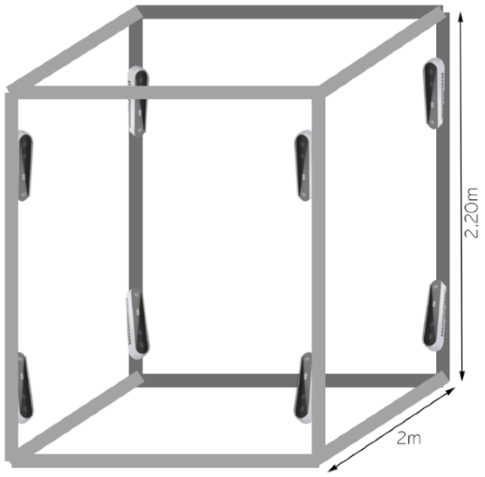

Figura 2.1: Estructura fija completa

Figura 2.2: Postes móviles.

De forma definitiva la estructura para el proceso de testeo está dando buenos resultados con una distancia de 1600 mm entre postes y el montaje es el que se puede ver a continuación (ver Figura 2.3)

Figura 2.3: Estructura de adquisición para proceso de testeo.

Para la versión *release* de la cabina se empleará el formato definido como fijo (ver Figura 2.1) pero con las medidas entre postes del que se han llevado a cabo los experimentos y se han obtenido buenos resultados, que son 1600 mm.

2.2 Cámaras

Aunque hace unos años no contábamos con mucha variedad de dispositivos de captura RGB-D de bajo coste (Villena Martínez, 2015), hoy en día ya contamos con un amplio abanico de dispositivos que ofrecen esta opción, incluso hay varios dispositivos móviles de alta gama que incorporan la tecnología de estéreo activo mediante proyección de patrón infrarrojo.

Como dispositivo de captura he elegido un modelo de cámara que salió al mercado el año pasado y la relación características/precio es de las mejores que he podido encontrar según se puede observar en la siguiente comparativa (ver Figura 2.4).

Nombre	**Precio €**	**Res.RGB**	**Res.Depth**	**FOVº**
Kinect	150	1280x1024	640x480	57x43
Kinect 2	199	1920x1080	512x424	70x60
Xtion2	263	2592x1944	640x480	74x52
Carmine	295	1280x960	640x480	54x45
RealSense D435	**165**	**1920x1080**	**1280x720**	**85x58**
RealSense D415	155	1920x1080	1280x720	69x42

Figura 2.4: Comparativa cámaras RGB-D

La cámara RealSense D435 es un dispositivo de última generación con una resolución notablemente superior respecto al resto y con un campo de visión que

también marca la diferencia. Por otro lado su precio es de los más bajos que podemos encontrar en la tabla.

Otro punto importante ha sido, que el tipo de estéreo activo que se realiza mediante el patrón proyectado, está pensado para poder trabajar con diversas cámaras en la misma escena y cada cámara proyecta patrones aleatorios cada vez que inicia una captura con la idea de que no interfieran en los posibles sensores situados en una cámara ubicada enfrente.

Por todas estas características este modelo será el que finalmente se utilice.

2.3 Hardware servidor

Dar forma a un equipo con la posibilidad de manejar el conjunto de cámaras, ser capaz de maniobrar con la información que estas entregan de forma eficiente y en un margen de tiempo reducido ha sido una tarea un tanto compleja debido a ciertos detalles.

Uno de los más laboriosos ha sido conseguir que el equipo fuera capaz de recibir de forma síncrona el flujo de información que entregan las 8 cámaras a máxima resolución, a 30 fps, sin perder datos y de forma estable. Por otro lado, el resto de elementos han sido seleccionados en base a los criterios expuestos en los siguientes puntos.

- Expansión USB 3.1

Las cámaras RealSense trabajan con el protocolo USB 3.1 Gen 1 que permite una velocidad de transferencia de 5Gb/s por cada puerto. Los equipos actuales ya incorporan este tipo de puertos en la placa base, por lo general 4, pero suelen estar multiplexados con una controladora. Con este detalle se permite la transferencia de datos a estas velocidades, pero se entiende que no se va a trabajar con los 4 puertos

con máximo flujo de datos a la vez, detalle que para este caso sí que es importante llevar a cabo. Además es necesario incorporar 4 puertos más, por lo que había que incluir otro dispositivo que permita llevar a cabo esta ampliación.

El escenario encontrado a la hora de elegir este dispositivo ha sido que la mayoría de tarjetas controladoras de USB 3.1 tenían el mismo problema que la placa base, eran tarjetas de 4 puertos pero con una controladora. Finalmente tuve que localizar un modelo que incorporase 4 puertos y 4 controladoras (una por puerto) y que fuese conectada al slot PCI-Express. No fue tarea fácil pero finalmente localicé un modelo que cumplía este requisito. (ver Figura 2.5 y Figura 2.6) (IOI Tech Corporation, n.d.) Con dos controladoras de este tipo hubo suficiente para conectar las 8 cámaras sin pérdidas en el flujo de datos y con una estabilidad aceptable.

Figura 2.5: Tarjeta USB 3.1 Gen1 PCIEx4.

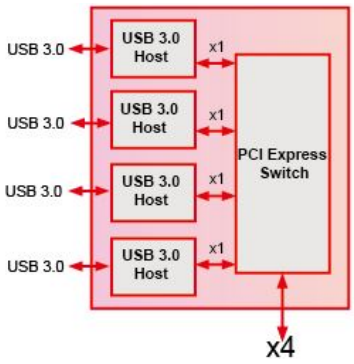

Figura 2.6: Esquema interno tarjeta con 4 controladoras.

- Cables USB

Cada cámara viene con su cable incluido en el embalaje, pero son cables de 100cm y esta longitud no sirve a la hora de distribuir las cámaras en el sistema.

La parte de la localización de un modelo de cable con la longitud adecuada y que cumpla con las necesidades de transmisión de datos de forma estable también ha sido una de las más complejas. Esto se ha debido a que, aunque habían cables que en sus especificaciones indicaban que si cumplían los requisitos, a la hora de emplearlos esto

no ha sido así, y solo aceptaban protocolo de USB 2.0 con la limitación que esto supone de reducir el flujo de datos del puerto a 60 MB/s.

Finalmente se localizó un modelo de la firma IVOLER que realmente cumplía con los estándares del USB 3.1.

- Procesador

En ciertas ocasiones se llevarán a cabo procesos aptos para ser paralelizados en diferentes núcleos por lo que era interesante contar con un procesador con como mínimo 8 de ellos. Interesaba que tuviese incorporado procesador de gráficos, puesto que aunque en un futuro se incorporará una tarjeta gráfica, estará reservada a la computación en paralelo de ciertos cálculos.

Otros requisitos importante eran que admitiese memoria RAM de última generación como es DDR4-2666Mhz con dos canales y que contase con una caché de acceso directo de mínimo 10Mb.

Otro requisito es que el procesador fuese de la firma Intel y dentro de los procesadores que ofertan, el que mejor se ajustaba a las necesidades era el modelo Intel core i7-9700K (ver Figura 2.7) que cuenta con las siguientes características:

- Cantidad de subprocesos 8
- Frecuencia básica del procesador 3,60 GHz
- Caché 12 MB SmartCache
- Tamaño de memoria máximo 128 GB
- Tipos de memoria DDR4-2666
- Cantidad máxima de canales de memoria 2
- Gráficos del procesador Intel® UHD Graphics 630
- Memoria máxima de video para gráficos 64 GB

Figura 2.7: Procesador Intel Core i7-9700K.

- Memoria

Para la memoria de acceso aleatorio se ha elegido la cantidad de 32GB, por ser esta una cantidad en la que se puede almacenar con cierta comodidad la cantidad de datos necesarios para el proceso de adquisición y no ser este un equipo en el que se va a trabajar de forma multi-tarea en el que se requiera memoria para diferentes procesos de forma simultánea.

Como el procesador cuenta con la posibilidad de gestionar dos canales de forma nativa, he optado por dos módulos de 16 Gb cada uno (ver Figura 2.8) de la firma Kingston y su gama HyperX con la que ya he trabajado anteriormente y la experiencia ha sido muy satisfactoria.

Figura 2.8: Memoria Kingstone HyperX Predator DDR4 2666Mhz 16Gb

- Almacenamiento

Para la parte de memoria no volátil se ha optado por dos discos duros.

Un disco Samsung SSD 860 Evo con interfaz SATA 6Gb y velocidades de lectura/escritura simétricas de 550MB/s (ver Figura 2.9) para el sistema operativo y

en el que en primera instancia se guardarán los modelos escaneados en el momento por temas de velocidad, con una capacidad de 512Gb.

Un disco Western Digital NAS Red SATA 6Gb 3Tb y 5600rpm (ver Figura 2.10) que será el encargado de almacenar la información relativa a los pacientes. En un futuro este disco estará formado por un raid en otro servidor externo.

Figura 2.9: Disco Samsung SSD 860 Evo.

Figura 2.10: Disco Western Digital NAS Red 3Tb.

- Placa base

Para unir todos estos componentes se ha elegido una placa base de la firma Asus, el modelo PRIME Z370-A II (ver Figura 2.11) por tener 3 slots PCIE x 16, dos para la expansión de puertos y uno para la futura gráfica. El resto de opciones son sencillas.

Figura 2.11: Placa base Asus Prime Z370-AII.

- Fuente de alimentación

Para la alimentación del equipo se ha elegido una fuente de la firma Corsair modelo RM 850x (ver Figura 2.12) con 850w que en un futuro sustentará el uso de la tarjeta

gráfica y el resto de componentes que podrán funcionar sin problemas de forma simultánea a máximo rendimiento.

Figura 2.12: Fuente Corsair RM850x.

2.4 Sistema operativo

La elección del Sistema operativo fue algo complejo puesto que hay múltiples opciones. Después de una primera criba quedaron 3 candidatos que serán detallados a continuación y finalmente se optó por utilizar Ubuntu en su versión 18.04LTS.

a. Windows Server 2016: Versión del sistema operativo de Microsoft que como principal punto débil cuenta con que es de pago. Por otro lado todas las librerías necesarias se encuentran disponibles para este sistema, pero la comunidad investigadora que las utiliza no está excesivamente extendida y no hay mucho apoyo comunitario al respecto.

b. CentOS: Es una versión pública del sistema operativo Red-Hat y, aunque es muy empleado en servidores web, datos, etc, para nuestro cometido no cuenta con excesivo apoyo por parte de la comunidad investigadora a la hora de solucionar configuraciones con las librerías que vamos a emplear.

c. Ubuntu: Es un sistema operativo de código abierto, gratuito, y tiene un buen servicio de actualizaciones. También cuenta con una comunidad de usuarios, tanto a nivel particular como a nivel investigador, muy activa.

Todas las librerías que se han empleado para llevar a cabo este proyecto cuentan con apoyo técnico para su uso en este sistema operativo.

Figura 2.13: Logotipo Ubuntu.

Ubuntu es un sistema operativo de código abierto para ordenadores. Este sistema es una distribución de Linux basado en la arquitectura Debian y concretamente he empleado la versión 18.04 LTS *Bionic Beaver* que fue lanzada el 26 de abril de 2018.

Al ser esta una versión de soporte extendido (LTS = Long Term Support) se nos garantiza que contaremos con actualizaciones de seguridad y soporte técnico durante un periodo de 5 años, es decir, hasta abril de 2023.

2.5 SDK Intel Realsense

Figura 2.14: Logotipo Intel Realsense.

Una de las tecnologías externas empleadas es el SDK (Software Development Kit) que proporciona Intel para sus dispositivos Realsense. En concreto para este proyecto he trabajado con la versión 2.0 del mismo y aunque solo se ha utilizado la versión para Linux y el lenguaje C++, es una librería que cuenta con versión para Windows y admite varios *wrappers* y lenguajes de programación como C, Python, Node.js API, ROS y LabVIEW.

Aparte incluye herramientas de calibrado y un visor que también se ha empleado, por ejemplo, en la automatización del calibrado tal y como se detalla en el capítulo 3.3. Se ha empleado esta librería en base al modelo de cámara elegido.

2.6 Meshlab

Figura 2.15: Logotipo Meshlab

Meshlab es un programa de código abierto que nos permite procesar y editar mallas triangulares en 3D. El mismo nos proporciona un conjunto de herramientas para editar, limpiar, inspeccionar, renderizar, texturizar y convertir mallas. Aparte de la edición de mallas ofrece diversas funciones entre las que destacan funciones para procesar datos *raw* producidos por herramientas y dispositivos de digitalización 3D y para preparar modelos para la impresión en 3D.

Meshlab también ofrece opciones de trabajo como servidor mediante scripts ejecutados a través de la línea de comandos, esta opción ha sido clave para la elección de este programa. En concreto se ha empleado la versión 2016.12-2 que es la más reciente.

2.7 PCL

Figura 2.16: Logotipo Point Cloud Library

La librería de nubes de puntos PCL (*Point Cloud Library*) es una librería de código abierto que contiene algoritmos para tareas de procesamiento de nubes de puntos y procesamiento de geometría 3D, como las que se producen en la visión tridimensional por ordenador. Esta librería contiene algoritmos para la estimación de características, reconstrucción de superficies, registro 3D, ajuste del modelo y segmentación. Está escrito en C++ y publicado bajo la licencia BSD. He seleccionado esta librería, ya que opciones como GTK, OpenMesh, CGAL, MFEM… aparte de no tener la cantidad de funciones con las que cuenta PCL, no cuentan con una comunidad de usuarios con tanto volumen y actividad como PCL.

2.8 OpenCV

Figura 2.17: Logotipo OpenCV.

OpenCV es una librería de visión artificial y edición de imagen libre que originalmente fue desarrollada por Intel. Es multiplataforma, y existen versiones para GNU/Linux, Mac OS X, Windows y Android. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos, calibración de cámaras, visión estéreo y visión robótica. El uso que se le va a dar a la librería va a ser desplazar la imagen un par de pixeles en cada uno de sus ejes. Se ha elegido esta librería para ello debido a la simplicidad de su uso y que ya se había trabajado con ella anteriormente.

3 CALIBRADO DEL SISTEMA

A lo largo de este capítulo se detallarán los procesos llevados a cabo para extraer los parámetros de calibrados intrínseco y extrínseco así como el proceso de automatización para la adquisición de la batería de imágenes para este último.

3.1 Intrínseco

Tal y como se detalla en los apartados 2.2 y 2.3, el calibrado intrínseco es una parte bastante importante para corregir las imperfecciones causadas por la lente.

Para llevar a cabo este proceso vamos a seguir la propuesta detallada por Intel (*Intel ® RealSense™ Depth Module D400 Series Custom Calibration*, 2019) en la que sigue los pasos que se detallan en el siguiente diagrama (ver Figura 3.1)

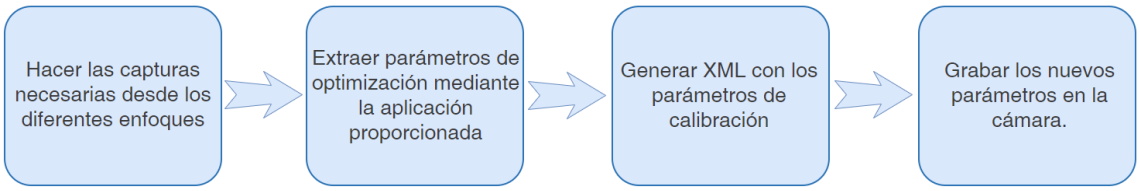

Figura 3.1: Proceso de calibrado intrínseco con la herramienta de Intel.

3.1.1 Captura de imágenes

El algoritmo implementado por parte de Intel en la aplicación de calibrado requiere un mínimo de seis muestras del objeto desde diferentes puntos de vista distribuidas según se indica en la siguiente imagen del interfaz (ver Figura 3.2)

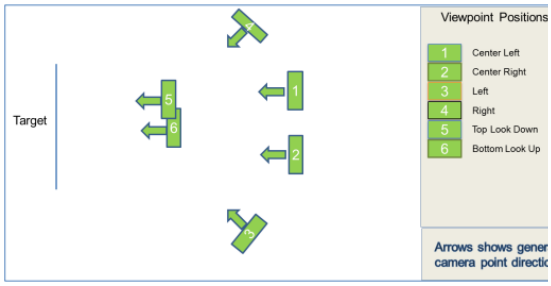

Figura 3.2: Esquema de distribución de posiciones de captura.

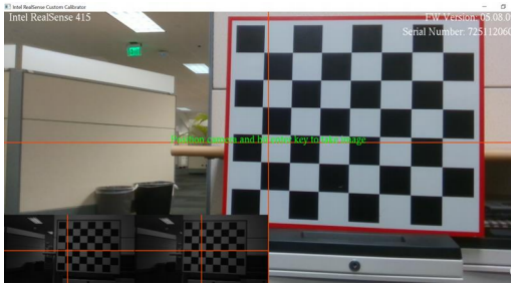

Figura 3.3: Interfaz de captura con patrón chessboard.

Como patrón objetivo para realizar el calibrado usaremos un patrón *chessboard* como el que se puede observar en la imagen anterior (ver Figura 3.3), que servirá al algoritmo empleado para identificar las imperfecciones en la imagen para posteriormente devolver los ajustes de corrección pertinentes.

3.1.2 Extracción de parámetros y generación de XML

En este punto proporcionamos al software la batería de imágenes capturadas y le solicitaremos que aproxime los valores de corrección para después exportarlos en formato XML.

3.1.3 Grabar parámetros en la cámara

Para finalizar el proceso de calibrado se graba el XML obtenido con los parámetros en el dispositivo mediante el software proporcionado por Intel. Estas cámaras contienen una memoria EEPROM interna para almacenar varios ajustes de captura y los parámetros de calibrado.

3.2 Extrínseco

El calibrado extrínseco tiene un papel muy importante tal y como se comenta en los apartados 1.2.2 y 1.2.3, para este proyecto se han llevado a cabo dos opciones ya implementadas previamente, un calibrado mediante el modelo de una esfera y un calibrado mediante el modelo de un cubo.

3.2.1 Calibrado mediante esfera

Hay diversos estudios relacionados con la calibración de un sistema mediante un objeto esférico (Minghao Ruan & Huber, 2014). Para llevar a cabo el proceso se ubica una esfera en una posición en la escena para hacer una captura y obtener distintas nubes de puntos, una por cada una de las cámaras, correspondiendo a una vista parcial de la esfera. Con esta información se selecciona aleatoriamente un sub-conjunto de puntos de la vista y se genera un sistema de ecuaciones que calcula el radio de una esfera que pase por los puntos del subconjunto seleccionado. Esta operación se realiza para diferentes subconjuntos. Por cada esfera generada para cada uno de los subconjuntos, mediante RANSAC (Random Sample Consensus) se comprueba el ajuste del resto de puntos no seleccionados al modelo de la esfera generado y se elige el centro del modelo de esfera que menos error muestre para el conjunto de puntos seleccionado (ver Figura 3.4).

Una vez se tiene este centro, se pone en común con el mismo proceso de las distintas vistas y se calcula la matriz de transformación para que este centro coincida.

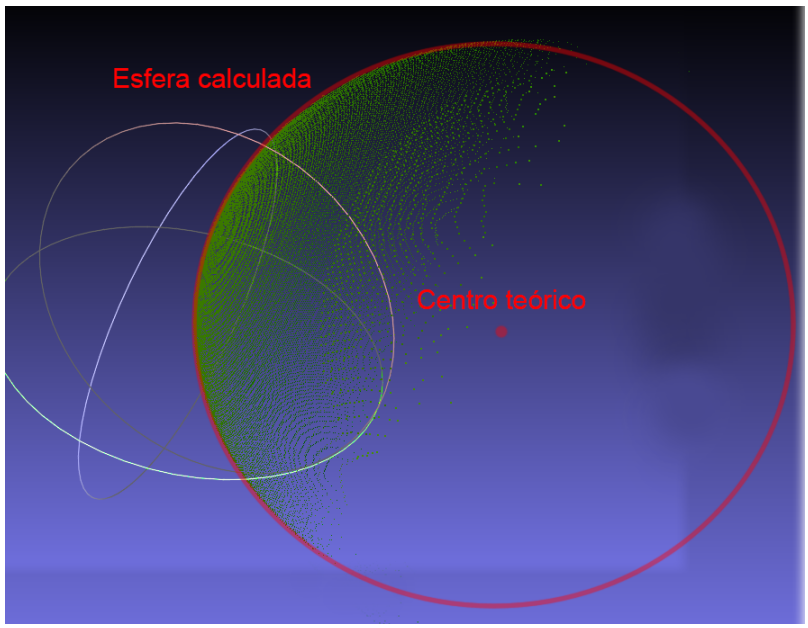

Figura 3.4: Esfera y centro teóricos calculados para calibrado con esferas.

Debido a que la relación de la captura no es lineal, este proceso puede dar matrices diferentes según en el punto de la imagen en el que esté situada la esfera, por lo que se realiza este proceso ubicando la esfera en distintos puntos de la captura y mediante *procrustes* se obtienen las matrices para cada una de las cámaras que minimicen el error en todo el conjunto de centros generados.

3.2.2 Calibrado mediante cubo

El proceso de calibrado mediante el modelo del cubo es similar al de la esfera. En primer lugar se calculan los planos teóricos de cada una de las vistas (ver Figura 3.5), para lo que es importante poder ver tres planos en cada una de ellas, se sabe que cada cara/plano forma un ángulo del 90º con los otros dos. Una vez se tiene el modelo de los tres planos y su posición se busca el encaje con los planos de la otra vista minimizando el error de los planos y esto ofrece una matriz de transformación (Saval-Calvo, Azorin-Lopez, Fuster-Guillo, & Garcia-Rodriguez, 2015).

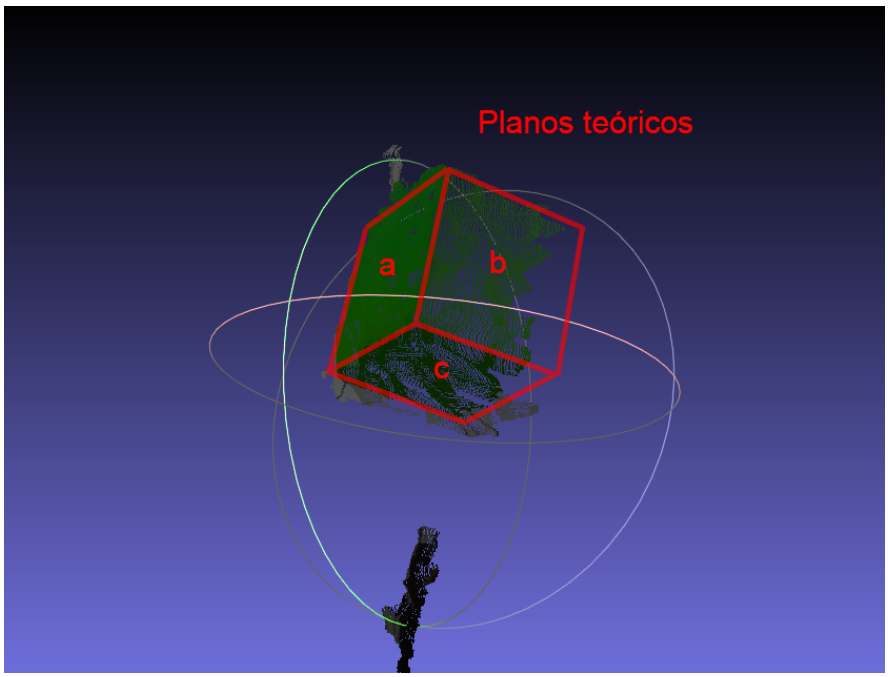

Figura 3.5: Planos teóricos calculados para una vista.

3.2.3 Estudio sobre el calibrado extrínseco

El objetivo a llevar a cabo estos dos tipos de calibrado es confirmar si contamos con ventajas al optar por el calibrado mediante el cubo en lugar del calibrado mediante esferas, al ser este primero un marcador que cuenta con mayor cantidad de puntos característicos y debido a esto cabe la posibilidad de que puedan obtenerse calibrados de la misma precisión que con el marcador esférico pero con un menor número de muestras. En el calibrado del cubo se llevan a cabo tres cálculos para identificar la figura, uno por cada plano en los que se encaja un modelo teórico, el del plano, frente al de la esfera que solo hace un cálculo. Por este motivo el calibrado del cubo es más robusto al ruido. Este estudio se ampliará en líneas de investigación futuras.

3.3 Automatización del calibrado

Ambos procesos de calibrado extrínseco implementados cuentan con el factor común de necesitar capturar un número bastante elevado de imágenes en distintas posiciones. En un origen el proceso era bastante tedioso puesto que había que colocar

el marcador en su posición y hacer la captura y así de forma repetitiva. Para aliviar esta tarea he desarrollado una función que lleva a cabo el número de capturas indicado por parámetro dejando un espacio de tiempo entre capturas que también se puede personalizar.

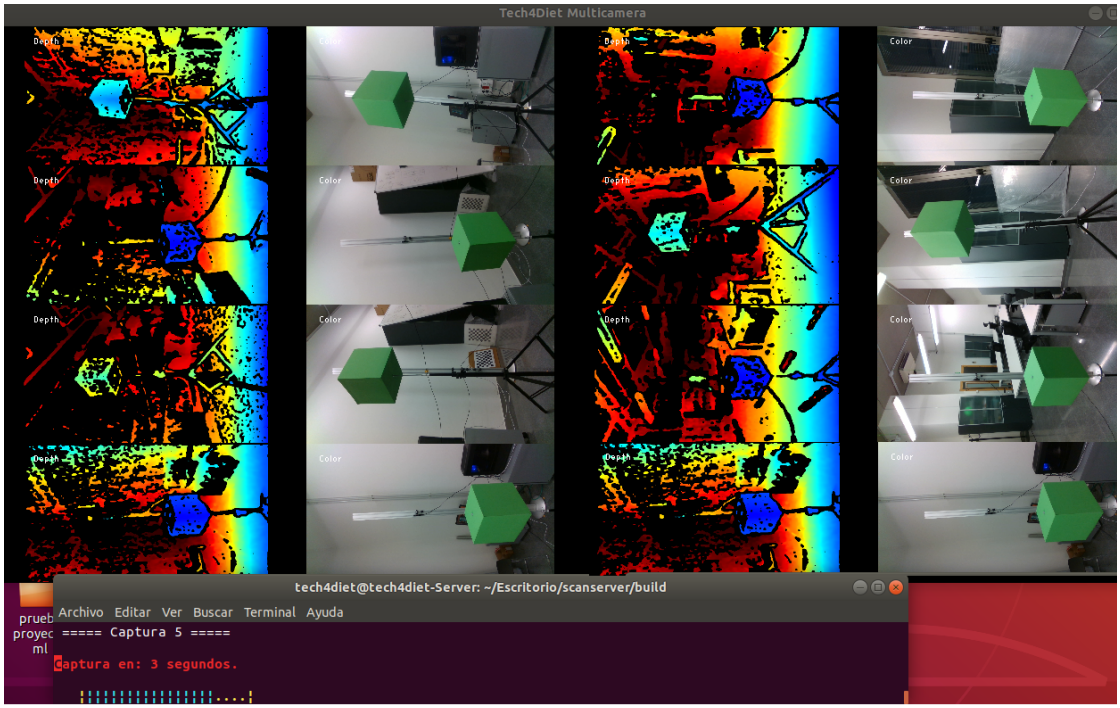

Figura 3.6: Ejecución del proceso de captura automatizado para el calibrado.

Como se puede observar en la imagen anterior (ver Figura 3.6) se muestra un visor en el que el usuario puede tener controladas las vistas de cada una de las cámaras, tanto la de profundidad como la de color, y una barra de progreso que indica los segundos restantes para la siguiente captura así como el índice de la captura que está haciendo actualmente con el fin de saber cuántas capturas restan para finalizar. Cada segundo de la cuenta atrás irá emitiendo un sonido. Su uso será detallado en el capítulo 4.5.3.

4 ENTORNO DE ADQUISICIÓN Y MODELADO

El proceso de adquisición cuenta con diversas etapas. En cada una de ellas se han testeado distintas propuestas que serán desarrolladas a lo largo de este capítulo. En un primer apartado se expone como se ha tratado la gestión de la captura de las cámaras mediante hilos, las opciones trabajadas para la sincronización de estas, etc… En el siguiente apartado se explican los pasos llevados a cabo para la gestión de las nubes de puntos, filtros, cálculo de normales, modificaciones sobre estas, etc… En el tercer apartado se ha desarrollado el proceso seguido para la generación de la malla a partir de la nube de puntos para a continuación, en el penúltimo apartado, detallar el proceso seguido para lograr el texturizado.

Finalmente para el último punto, se ha hecho una recopilación del proceso seguido para enlazar todas estas etapas con el fin de que funcionen de forma sincronizada.

4.1 Adquisición de la nube de puntos

Para la gestión de las cámaras y la adquisición de las nubes de puntos por cada una de ellas se ha desarrollado un sistema de hilos. La función `rs2::context().query_devices()` que proporciona la librería realsense2 incluida en el SDK de Intel me da la información de cuantas cámaras hay conectadas en el equipo. Esta información la he usado como valor para iterar con un bucle que con la función:

`query_sensors().front().get_info(RS2_CAMERA_INFO_SERIAL_NUMBER);` extrae el número de serie de la cámara y que posteriormente uso para lanzar un hilo que gestiona la cámara hasta que finaliza la adquisición. Cada uno de estos hilos se acumula en un vector de hilos con la siguiente instrucción: `threads_vector.emplace_back([serial, cfg, session_folder, once]() { save_from_camera(serial, cfg, session_folder, once); });`

4.1.1 Sincronización cámaras

Durante la extensión del subcapítulo detallaré los dos sistemas con los que se han trabajado para controlar la sincronización de las cámaras, siendo estos, por software mediante semáforos y generando una estructura Master-Slave empleando una interconexión hardware.

4.1.1.1 Semáforos

Según se ha implementado, después del proceso de inicialización de las cámaras cada uno de los hilos queda a la espera en un bucle mediante un semáforo que indica en qué momento hay que hacer la captura.

Los semáforos actúan de forma similar a una barrera, y aseguran que todos los hilos comienzan a capturar prácticamente al mismo instante, y que no finalizan la adquisición del frame hasta que no han terminado el resto de cámaras, para evitar que algunos hilos avancen más que otros.

La gestión cuenta con dos flags, `do_capture` y `do_exit`, y una variable de condición `cv` que coordina la comunicación entre todos los hilos. Cada hilo queda a la espera en su debido momento con `cv.wait();` y son invocados a iniciar su gestión de forma síncrona con `cv.notify_all();`, el código del proceso de control de los flags que capturan, cierran y notifican a los hilos es el siguiente:

```
do_capture = true;
cv.notify_all();
do_exit = true;
cv.notify_all();
```

4.1.1.2 Master-Slave

Este modelo de cámara de Intel posee la característica de poder crear una estructura de sincronización Master-Multi Slave por hardware (Grunnet-Jepsen et al., n.d.) a través de su puerto de comunicación como se puede observar en la siguiente imagen (ver Figura 4.1). Esta función lleva a cabo la sincronización de captura de fotogramas entre los distintos equipos conectados y coordinados por uno de ellos, definido como master.

Figura 4.1: Puerto de conexión para sincronización RealSense.

No obstante finalmente este sistema no se ha empleado puesto que se ha llegado a la conclusión que para obtener una captura estática, es suficiente con la precisión que proporciona el sistema desarrollado en el punto anterior. Mediante la gestión por semáforos conseguimos hacer un ajuste grueso de la sincronización entre dispositivos, y a través del hardware conseguimos una sincronización precisa, ya que la cámara maestra envía un pulso a las esclavas para que la adquisición del *frame* se realice en el mismo instante, por lo que el *delay* máximo que puede haber es lo que tarde la señal de una cámara a otra, qué es mínimo.

4.2 Preprocesamiento y registro de nubes de puntos

A lo largo del capítulo se detallará el proceso llevado a cabo con la nube de puntos, los pasos de su preprocesado y la generación del registro.

4.2.1 Preprocesamiento

En este capítulo serán expuestos los pasos de preprocesado que se han llevado a cabo con la nube de puntos. Truncado de esta, la aplicación de diversos filtros y el cálculo de las normales en cada uno de los puntos.

4.2.1.1 Truncado

El primer paso que se realiza nada más capturar la nube de puntos es llevar a cabo un truncado en el eje Z de la misma. Actualmente la disposición de los postes de la cabina genera una diagonal de 2200 mm, lo que sitúa el centro en 1100mm. Si tenemos en cuenta que el cuerpo que vayamos a escanear va a estar desde el centro hacia la cámara, podríamos eliminar directamente los vértices que excedan este punto central.

Un valor que ha dado buenos resultados para este ajuste es truncar los puntos que excedan 1150 mm, es válido si tenemos en cuenta que no deben de haber más puntos con vista frontal (las vistas opuestas se captarán por la cámara enfrentada) hasta el siguiente poste, puesto que la cabina debe de estar vacía, por lo tanto truncar hasta 2000 mm no debería de ejercer ninguna diferencia en la información truncada.

No obstante en las siguientes imágenes podemos observar la diferencia entre una captura de una vista truncada a 1150 mm (ver Figura 4.2) y otra sin truncar (ver Figura 4.3). Una de las cosas que más destaca es los 921.000 vértices que con los que cuenta la nube sin truncar, frente a los 65.000 a los que se ha visto reducida la nube truncada.

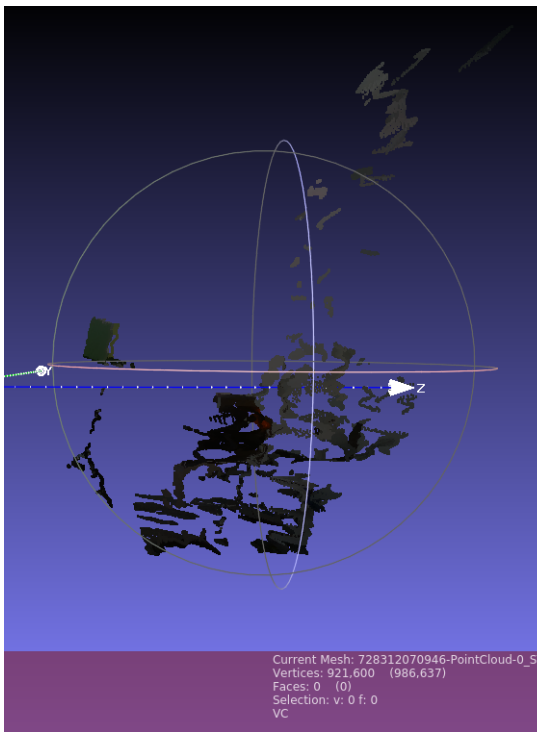

Figura 4.2: Nube de puntos sin truncar.

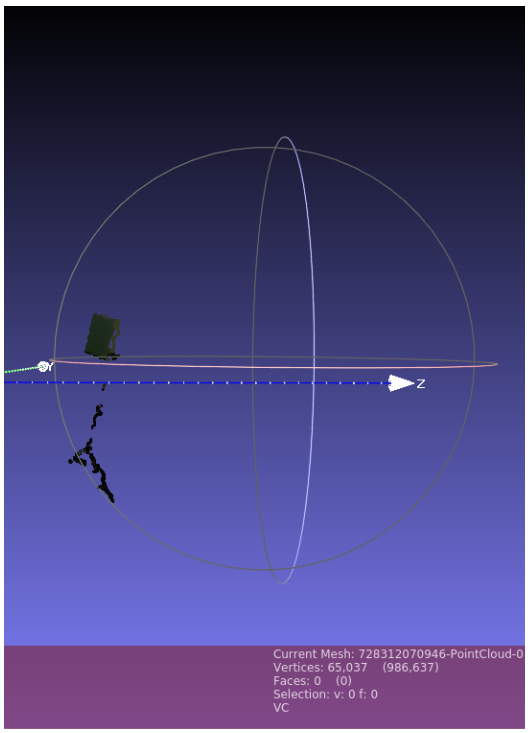

Figura 4.3: Nube de puntos truncada a 1500mm.

4.2.1.2 Filtros

En este capitulo se van a desarrollar las pruebas e implementaciones que se han llevado a cabo para filtrar la nube de puntos. Los filtros aplicados han sido tres, el filtro de mediana, el filtro bilateral y el Statistical Outlier Removal.

4.2.1.2.1 Mediana

El filtro de mediana se encuentra implementado en la librería PCL (PCL Team, 2013a) y es uno de los filtros de procesamiento de imágenes más sencillos y extendidos. Es un filtro que funciona bien con píxeles sueltos con valores extremos, no reduce el contraste en todos los pasos de la función, cosa que en parte sí sucede con los filtros basados en promedios, y es robusto con los valores atípicos. Además, es un filtro muy eficiente, ya que requiere una sola pasada sobre la nube. Consiste en una ventana móvil de tamaño fijo que reemplaza el punto del centro con la mediana de los valores de los puntos de la ventana.

Es importante destacar que este algoritmo filtra sólo la profundidad (componente z) de nubes de puntos organizadas.

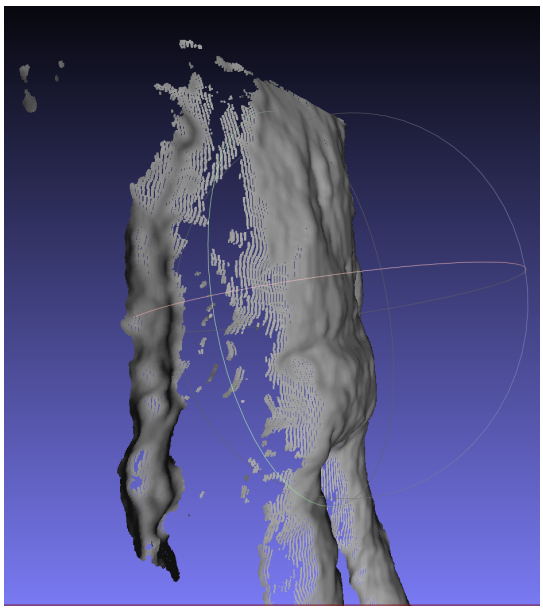

Figura 4.4: Vista trasera sin filtro de mediana.

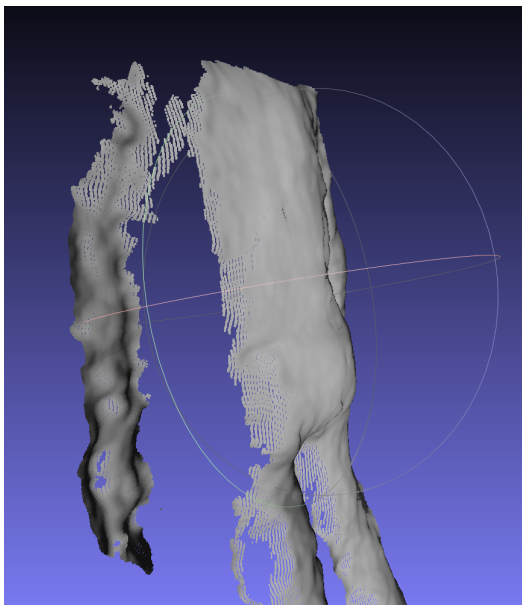

Figura 4.5: Vista trasera con filtro de mediana.

Como tamaño de ventana he decidido fijar un valor de 9, por ser con el que mejores resultados he obtenido. Como se puede ver en las imágenes anteriores (ver Figura 4.4 y Figura 4.5) este filtro lleva a cabo un suavizado de las superficies suavizando las diferencias en el eje z. A su vez también elimina información de los bordes, por considerarla como una variación elevada al haber poca densidad en poco espacio.

4.2.1.2.2 Bilateral

Este filtro que se encuentra implementado en la librería PCL (PCL Team, 2019a) suaviza sin perder nitidez en zonas importantes como bordes o cuando hay grandes curvaturas y lo hace mediante una combinación no lineal de valores de las zonas cercanas. El método es no iterativo, local y simple, y utiliza el canal de intensidad para llevar a cabo su tarea.

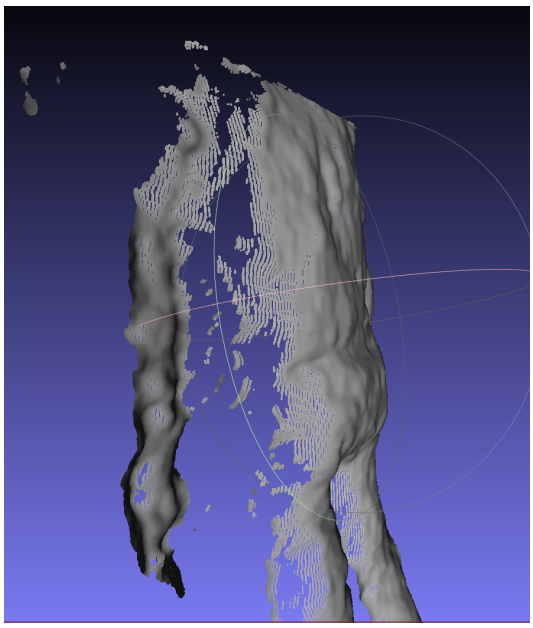

Figura 4.6: Vista trasera sin filtro bilateral.

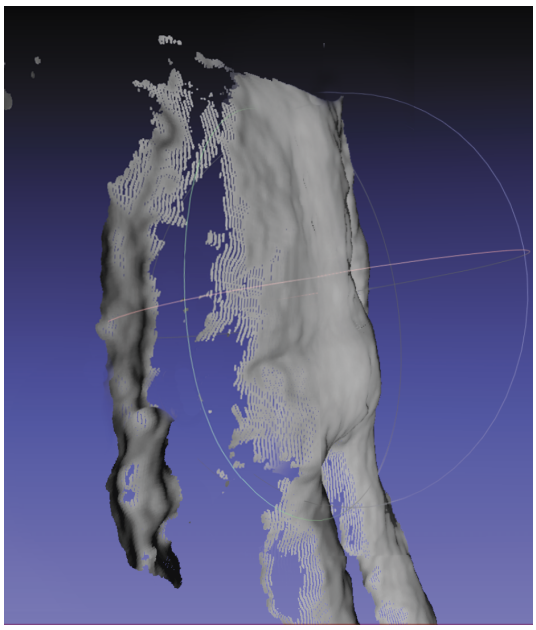

Figura 4.7: Vista trasera con filtro bilateral.

Como parámetros definitivos después de las pruebas realizadas he elegido 1.0 para el tamaño medio de la ventana del filtro bilateral gaussiano y 0.2 como desviación estándar.

Como se puede observar en las imágenes anteriores (ver Figura 4.6 y Figura 4.7), este filtro suaviza los desniveles del eje z pero no penaliza tanto en los detalles como flancos y demás lugares donde hay cambios más bruscos en el cambio de la información correspondiente a este plano.

4.2.1.2.3 Statistical Outlier Removal

Este filtro implementado en la librería PCL (PCL Team, 2013) se encarga de eliminar ruidos en los bordes o datos atípicos mediante estadísticas de vecindad, para hacerlo itera a través de toda la nube dos veces. Durante la primera iteración calcula la distancia promedio que cada punto tiene a sus vecinos k más cercanos. A continuación, se calcula la media y la desviación estándar de todas estas distancias para determinar un umbral de distancia. Durante la siguiente iteración, los puntos se

clasifican como *inlier* u *outlier* si la distancia media de su vecino está por debajo o por encima de este umbral, respectivamente.

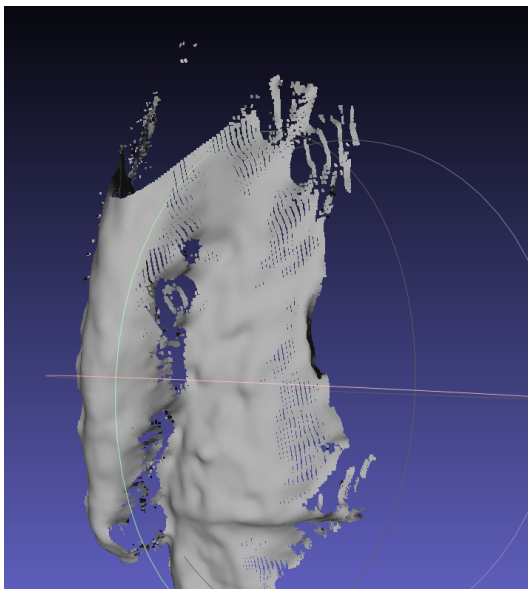

Figura 4.8: Vista lateral sin filtro SOR aplicado.

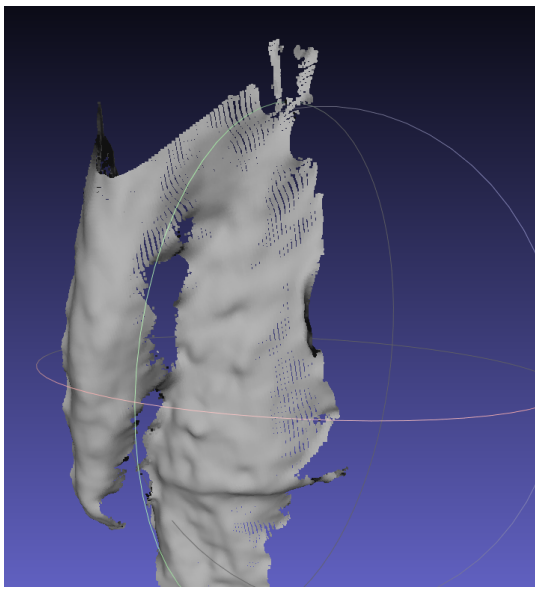

Figura 4.9: Vista lateral con filtro SOR aplicado.

Como se puede observar en las imágenes anteriores (ver Figura 4.8 y Figura 4.9) la aplicación del filtro elimina el ruido producido por el sistema de adquisición en la profundidad de los bordes de las nubes de puntos, llevando a cabo una mejora de precisión en estas zonas.

4.2.1.3 Cálculo de normales

Las normales de los vértices son propiedades importantes en ciertas superficies y son empleadas para distintos cometidos como aplicar fuentes de luz de forma correcta, generar sombras, aplicar ciertos efectos, y para lo que se van a usar en este caso que es aplicar filtros.

Una vez tenemos una superficie geométrica, suele ser trivial inferir la dirección de la normal en un cierto punto de la superficie como en vector perpendicular de la superficie en ese punto. Sin embargo, dado que los conjuntos de datos o nubes de puntos que adquirimos representan un conjunto de muestras de puntos en la superficie

real, existen dos posibilidades de aproximación. O bien obtener la superficie subyacente a partir del conjunto de datos de la nube de puntos adquirida, utilizando técnicas de malla superficial, y luego calcular las normales de la superficie a partir de la malla. O usar aproximaciones para inferir las normales de la superficie a partir del conjunto de datos de la nube de puntos.

Aunque en la actualidad existen muchos métodos de estimación normales diferentes, el que voy a emplear por estar implementado en la librería PCL está formulado de la siguiente manera (Radu Bogdan Rusu, n.d.). El problema de determinar la normal en un punto de la superficie se aproxima por el problema de estimar la normal de un plano tangente a la superficie, que a su vez se convierte en un problema de estimación de ajuste del plano en base a los mínimos cuadrados.

La solución para estimar la normal de una superficie se reduce por lo tanto a un análisis de los vectores propios y valores de una matriz de covarianza creada a partir de los vecinos más cercanos del punto de consulta. Más específicamente, para cada *Pi*, generamos la matriz de covarianza *C* como indica en la siguiente imagen (ver Figura 4.10).

$$\mathcal{C} = \frac{1}{k}\sum_{i=1}^{k} \cdot (\boldsymbol{p}_i - \overline{\boldsymbol{p}}) \cdot (\boldsymbol{p}_i - \overline{\boldsymbol{p}})^T,\ \mathcal{C} \cdot \vec{\mathbf{v}_j} = \lambda_j \cdot \vec{\mathbf{v}_j},\ j \in \{0, 1, 2\}$$

Figura 4.10: Fórmula para el cálculo de la Matriz de Covarianza.

Donde k es el número de puntos vecinos considerados para *Pi* y $\overline{P}$ representa el centroide 3D de los vecinos más cercanos, $\boldsymbol{\lambda j}$ es el j-ésimo valor propio de la matriz de covarianza, y $\vec{\mathbf{V}}\mathbf{j}$ el j-ésimo vector propio.

En general, debido a que no existe una forma matemática de resolver el signo de la normal, su orientación calculada mediante el análisis de componentes principales es

ambigua, y no está orientada consistentemente en relación a todo el conjunto de vértices que componen la nube de puntos.

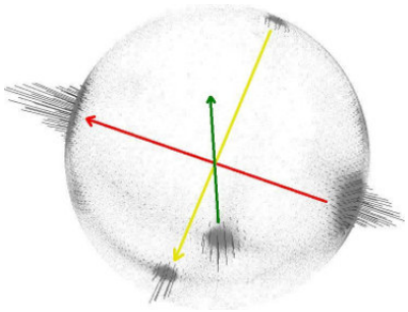

Figura 4.11: Esfera con normales calculadas sin tener en cuenta el punto de vista.

En la imagen anterior (ver Figura 4.11) se puede ver una esfera con todas las normales de la nube de puntos representadas. Si tenemos en cuenta que los conjuntos de datos han sido adquiridos desde un único punto de vista, las normales deberían estar presentes sólo en la mitad de la esfera. Sin embargo, debido a la inconsistencia de la orientación, el cálculo se extiende por toda la esfera.

La solución al problema es trivial si sabemos, como es nuestro caso, el punto de vista *Vp* a la hora de llevar a cabo la captura. Para orientar todas las normales hacia el punto de vista, necesitan satisfacer la ecuación que podemos ver a continuación (ver Figura 4.12)

$$\vec{n}_i \cdot (\mathrm{v}_p - \boldsymbol{p}_i) > 0$$

Figura 4.12: Ecuación para eliminar las normales no orientadas al punto de vista.

Respetando esta fórmula obtenemos el siguiente resultado (ver Figura 4.13), en el que se puede ver que todas las normales de los vértices de la figura anterior se han orientado consistentemente hacia el punto de vista.

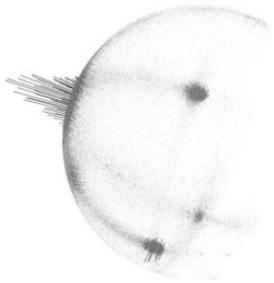

Figura 4.13: Esfera con normales calculadas teniendo en cuenta el punto de vista.

Como ya se ha comentado, la normal en un vértice necesita ser estimada mediante el apoyo circundante de los vecinos del vértice (también llamado k-vecinos).

La cantidad de vecinos a tener en cuenta es una cuestión de extrema importancia y constituye un factor limitante en la estimación de la representación de una característica puntual. En la siguiente imagen (ver Figura 4.14) se representan los efectos de seleccionar una escala más pequeña frente a una escala más grande. La parte izquierda de la imagen representa un radio correcto, con las superficies normales estimadas aproximadamente perpendiculares para las dos superficies planas y los pequeños bordes visibles en toda la tabla. Sin embargo, si el radio de vecinos propuesto es demasiado grande como en la imagen de la parte derecha, tenemos que el conjunto de vecinos es mayor que los puntos de cobertura de las superficies adyacentes, y las representaciones de las características de los puntos estimados se distorsionan, con normales de superficie rotada en los bordes de las dos superficies planas, bordes con ruido y con la consecuente supresión de los detalles finos.

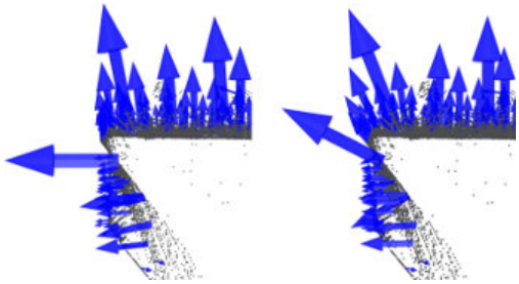

Figura 4.14: Normales calculadas con un radio adecuado y con radio no adecuado.

La librería PCL ofrece una implementación paralelizada mediante OMP de este algoritmo y es el que he utilizado de la siguiente forma:

```
NormalEstimationOMP<PointNormal,Normal> ne;
search::KdTree<PointNormal>::Ptr tree(new search::KdTree<PointNormal>());
PointCloud<Normal>::Ptr cloud_normals (new PointCloud<Normal>);
ne.setNumberOfThreads (sysconf(_SC_NPROCESSORS_ONLN));
ne.setInputCloud(clouds[i]);
ne.setSearchMethod(tree);
ne.setRadiusSearch(cfg.normalRadius);
ne.setViewPoint (0, 0, 0);
ne.compute(*cloud_normals);
```

Como estructura de búsqueda para los vecinos más cercanos he utilizado una estructura de árbol la cual se completa con la nube aportada y permite ahorrar tiempo de búsqueda a la hora de seleccionar los vecinos de un vértice.

Para el punto de vista he asignado el punto origen x=0,y=0,z=0 que es en el que se ubica la cámara en la nube de puntos obtenida.

Y por último el parámetro de radio viene establecido en el archivo de configuración. Para elegir este parámetro no ha habido que hacer excesivos testeos puesto que el tipo de objeto escaneado no consta de zonas excesivamente críticas por haber mucha densidad de vértices, no haber cantos vivos y ser, en su mayoría, zonas suavizadas. Un valor que ha dado buenos resultados ha sido un radio de 0,01 (ver Figura 4.15). Valores por encima han dado resultados similares, pero con mayor tiempo de cómputo y con valores inferiores, las normales perdían la forma de la superficie.

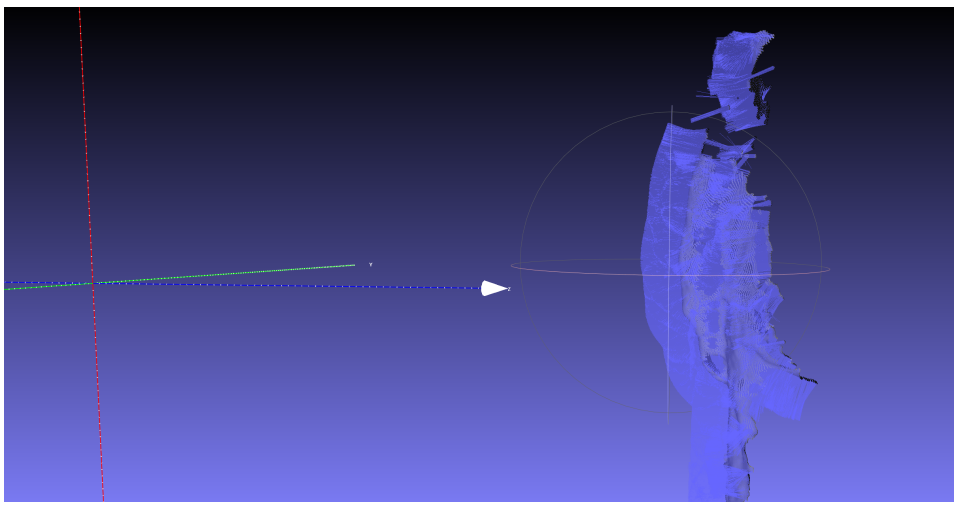

Figura 4.15: Normales calculadas orientadas hacia el punto de vista.

4.2.2 Registro

A lo largo de esta sección se tratarán los pasos desarrollados para conseguir el registro a partir de las diversas nubes de puntos, en primer lugar se le aplicarán las transformaciones y posteriormente se unificarán en una única nube de puntos.

4.2.2.1 Transformaciones

Una vez alcanzado este punto contamos con ocho nubes de puntos procedentes de cada una de las cámaras 3D convenientemente filtradas y con las normales calculadas. El siguiente paso antes de poder unirlas en un único espacio 3D es aplicar las matrices de transformación T obtenidas del calibrado extrínseco, el cual se ha detallado en el capítulo 3.5, para situarlas cada una en su posición y orientación correspondiente. Cada cámara tiene su propio sistema de referencia con el origen (0,0,0) ubicado en sí misma, con los parámetros extrínsecos se le aplicará la transformación a cada una de las nubes para unificar su referencia.

Figura 4.16: Transformación de nube para unificar sistema de coordenadas.

Como se puede ver en la imagen anterior, (ver Figura 4.16) al rotar 90º la nube de puntos obtenida por la cámara 2, se integran las dos nubes a un único sistema de coordenadas con origen en la cámara 1. En las siguientes imágenes (ver Figura 4.17 y Figura 4.18) se puede observar cómo se ha producido la transformación en una de las vistas para encajar con la anterior.

Figura 4.17: Vista de dos nubes de puntos con sistema de coordenadas propio.

Figura 4.18: Vista de la nube de puntos transformada en base al calibrado.

El proceso llevado a cabo es iterar cada nube con su matriz obtenida del archivo de configuración, procedente del calibrado previo. Una de las nubes no sufrirá transformación ya que tendrá una matriz de identidad. Esta cámara será el punto de origen. El código para este paso ha quedado de la siguiente manera:

```
std::vector<PointCloud<PointNormal>::Ptr> cloudsTransformed;
for(int i=0;i<clouds.size();++i){
       Eigen::Matrix4f transform = Eigen::Matrix4f::Identity();
       std::vector<float> temp = cfg.matrices[i];
       transform (0,0) = temp[0];
       transform (0,1) = temp[1];
       transform (0,2) = temp[2];
       transform (0,3) = temp[3];
       transform (1,0) = temp[4];
       transform (1,1) = temp[5];
       transform (1,2) = temp[6];
       transform (1,3) = temp[7];
       transform (2,0) = temp[8];
       transform (2,1) = temp[9];
       transform (2,2) = temp[10];
       transform (2,3) = temp[11];
       PointCloud<PointNormal>::Ptr transformed_cloud(new PointCloud<PointNormal>());
       transformPointCloudWithNormals (*clouds[i], *transformed_cloud, transform);
       cloudsTransformed.push_back(transformed_cloud);
}
```

4.2.2.2 Unificar nubes

Cuando ya se han llevado a cabo las transformaciones en cada una de las nubes el último paso que se hará sobre ellas es unificar las 8 nubes en una nube de puntos

completa. El proceso es sencillo, un bucle itera las nubes y va acumulando sus vértices en una nube global como se puede ver a continuación:

```
PointCloud<PointNormal> unified_cloud;
for(int i=0;i<clouds.size();++i){
       unified_cloud += *clouds[i];
}
```

En la siguiente imagen se puede observar un ejemplo del resultado esperado de la nube de puntos unificada (ver Figura 4.19)

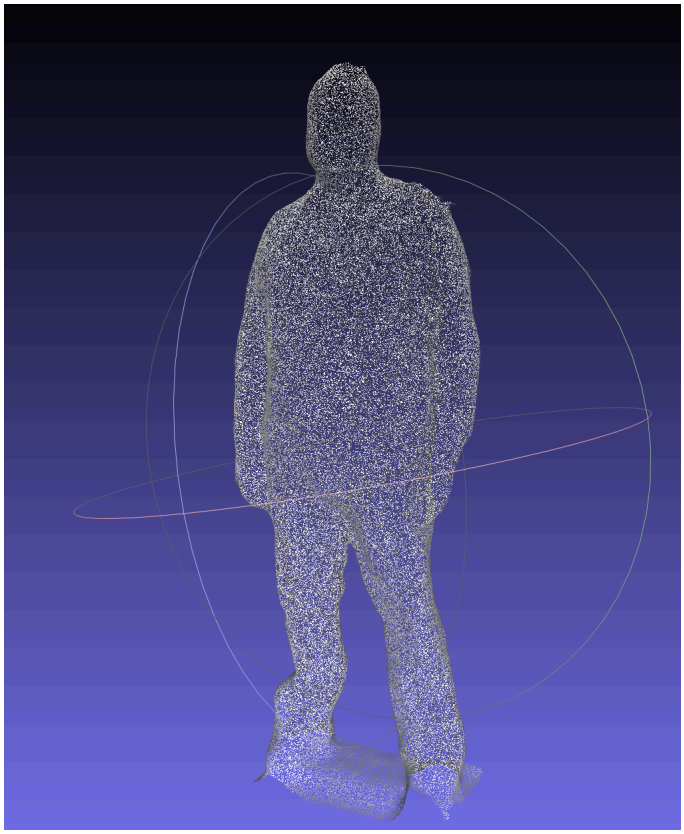

Figura 4.19: Registro esperado a partir de las nubes capturadas.

4.3 Generación de malla

En esta sección se detallará el proceso de generar la malla en base a la nube de puntos completa de la que se dispone en este momento. Para ello se han llevado a cabo distintas pruebas con diferentes algoritmos que realizan este proceso, concluyendo finalmente en utilizar el algoritmo de Poisson debido a los motivos que expondré a lo largo de la sección.

4.3.1 Estudio tipos de algoritmos

Como ya se avanzó en el apartado del estado del arte se han llevado a cabo pruebas con tres algoritmos de los más empleados como son Greedy Projection, Marching Cubes y Poisson, habiendo testeado en cada uno de ellos diversos parámetros.

4.3.1.1 Greedy projection

La triangulación greedy es un método para calcular una triangulación poligonal o una triangulación de conjunto de puntos utilizando un esquema greedy, que añade bordes uno por uno a la solución en estricto orden creciente por longitud, con la condición de que un borde no pueda cortar un borde previamente insertado. En función del orden seguido, la superficie que encierra el polígono creado (su normal) queda orientada hacia el exterior o el interior. Como se puede ver en el resultado (ver Figura 4.20) este algoritmo implementado en la librería PCL (PCL Team, 2019) deja muchos polígonos orientados hacia el interior, lo cual queda representado como un hueco.

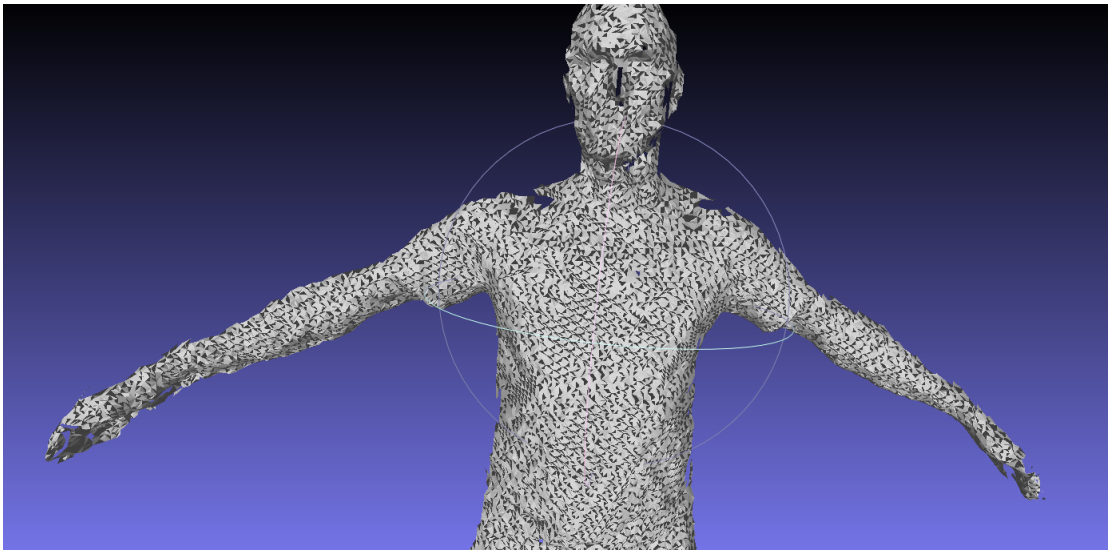

Figura 4.20: Malla resultante al aplicar Greedy Projection a la nube de puntos.

4.3.1.2 Marching Cubes

El objetivo de este algoritmo implementado en la librería PCL (PCL Team, 2019) es dividir la nube de entrada en un conjunto discreto de cubos. Asumiendo un filtro

de reconstrucción lineal, cada cubo, que contiene una parte de una determinada zona de la superficie, puede ser fácilmente identificado porque los valores de la muestra en los vértices del cubo deben incluir el valor de superficie objetivo. Para cada cubo, que contiene una sección de la superficie, se genera una malla triangular que se aproxima al comportamiento de una interpolación en el interior del cubo. El resultado de llevar a cabo el mallado de una vista frontal es el siguiente (ver Figura 4.21):

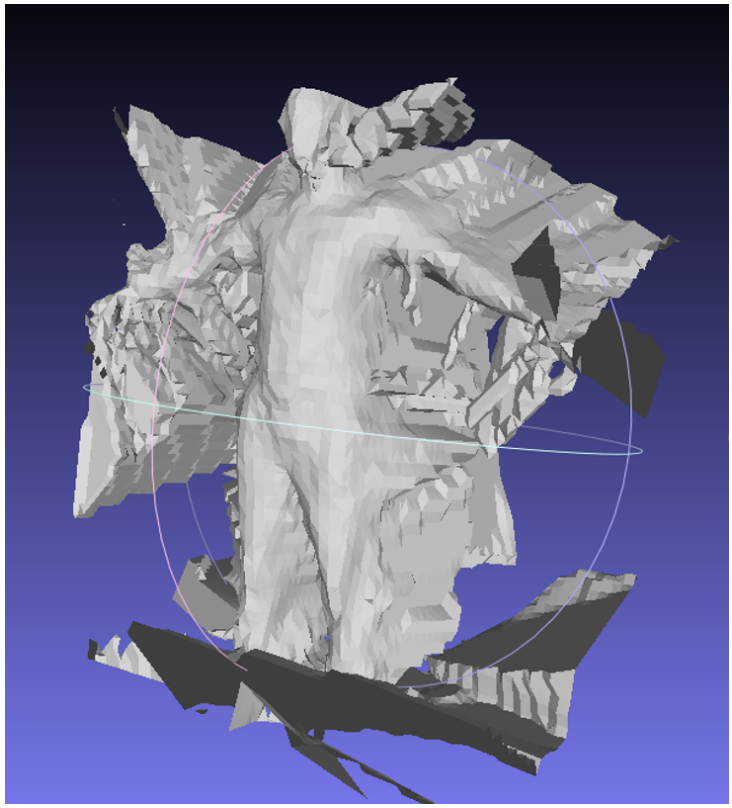

Figura 4.21: Resultado malla generada con algoritmo Marching Cubes.

4.3.1.3 Poisson

Poisson es un algoritmo de reconstrucción de superficies que se encuentra implementado en PCL (PCL Team, 2019d) y que consiste en encontrar una función que evalúe como mayor que 0 los puntos en el interior de la superficie y como menor los puntos en el exterior. Esta función se puede encontrar ya que existe una relación entre la orientación de los puntos y la propia función.

Específicamente, el gradiente de la función que se trata de buscar es un campo vectorial con valor 0 en todos los puntos excepto en aquellos próximos a la superficie donde toma el valor de las normales orientadas hacia el interior.

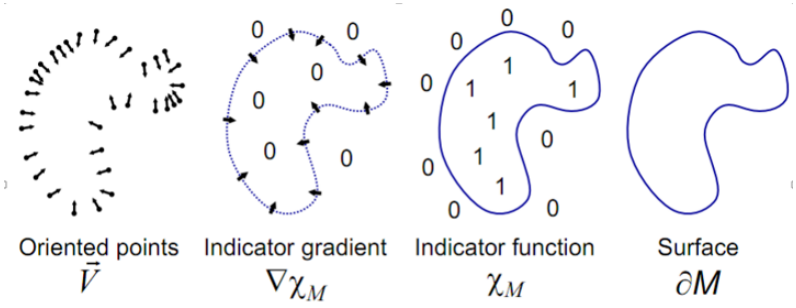

Figura 4.22: Funcionamiento algoritmo Poisson en dos dimensiones.

En la imagen anterior (ver Figura 4.22) se pueden observar los pasos que sigue el algoritmo Poisson en un espacio de dos dimensiones. Para nuestro caso es extrapolar el mismo proceso, pero en lugar de emplear una línea usar un plano. Resultado de esto, con un modelo de nube de puntos de una vista frontal, obtenemos el siguiente resultado (ver Figura 4.23) en el que el plano aproxima la nube de puntos por la parte frontal, pero no termina de cerrar la parte trasera al no encontrar una relación cercana.

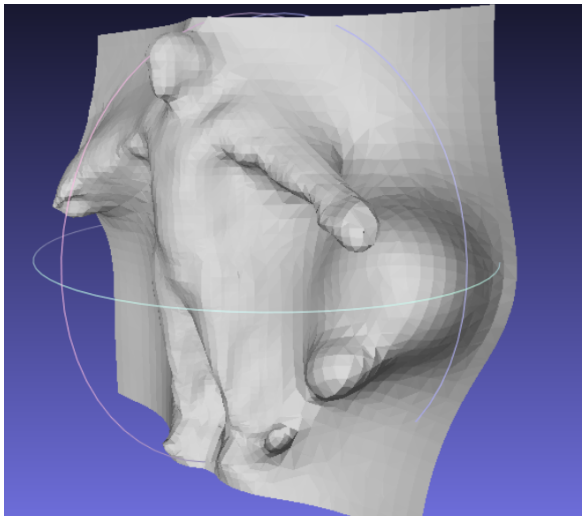

Figura 4.23: Poisson con una vista frontal.

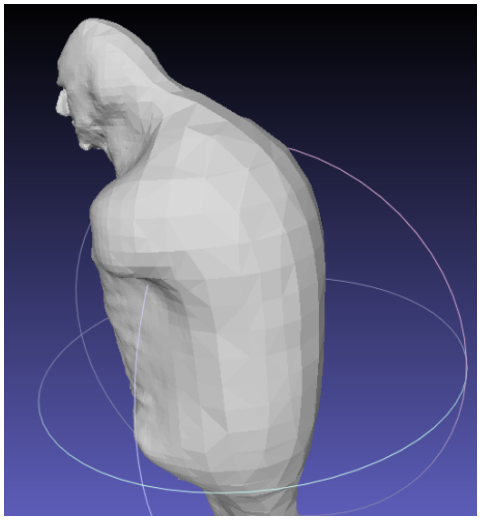

Figura 4.24: Poisson con vista frontal y lateral.

Sin embargo, como se puede ver en la imagen de la derecha (ver Figura 4.24) al añadir una nube de puntos lateral, el algoritmo ya encuentra la relación y cierra el hueco trasero hasta el lateral opuesto.

Para este algoritmo se puede ajustar un parámetro que se define como profundidad, y es directamente proporcional a la cantidad de detalle que tiene la malla con la que tratamos de envolver la nube de puntos. Finalmente el resultado de este algoritmo es bastante bueno (ver Figura 4.25) con una profundidad de 9, que encuentra un equilibrio entre detalle aceptable y velocidad de cómputo no muy elevada.

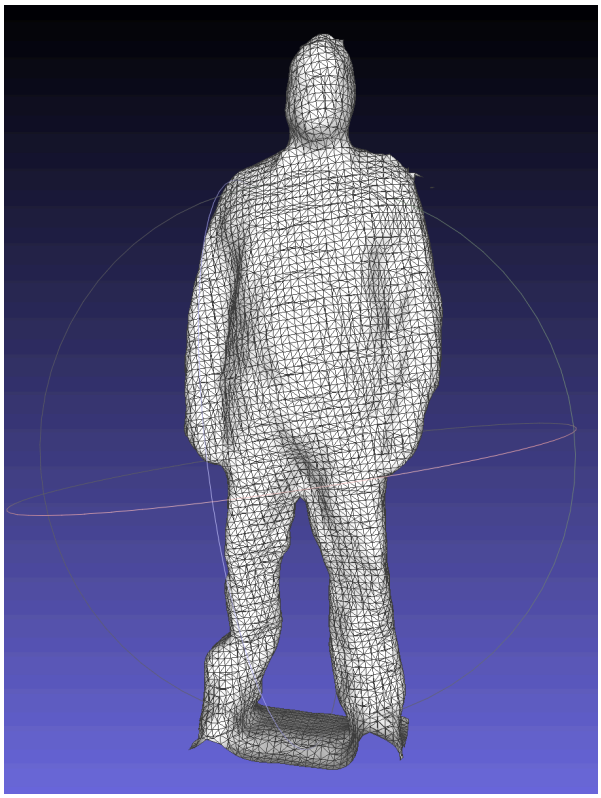

Figura 4.25: Mallado generado con algoritmo Poisson.

4.4 Proyección de rasters y generación de textura

Para realizar el texturizado se ha empleado un método implementado en el programa Meshlab, denominado "Parameterization + texturing from registered rasters". Este método genera parámetros de la malla en relación a sus vértices y genera

la textura en base a la proyección de las diferentes imágenes teniendo en cuenta la posición de las distintas cámaras. A continuación detallaré el funcionamiento del algoritmo empleado, como he llevado a cabo un testeo de forma sintética y finalmente algunos detalles significativos de la implementación.

4.4.1 Funcionamiento del algoritmo

Con el objetivo de proyectar los rasters de las distintas cámaras de las que disponemos sobre los polígonos que forman la malla, este algoritmo segmenta la malla en base a sus vértices y busca la correspondencia de estos sobre las distintas vistas valiéndose de las normales. Una vez seleccionada la mejor opción para elegir que raster proyectar sobre el polígono en cuestión, el algoritmo traza rayos virtuales (ver Figura 4.26) que van desde el punto del polígono hasta la posición de la cámara, atravesando el plano del raster, en el que impacta en un pixel, y esto genera una correspondencia entre el punto de la malla y el pixel del raster.

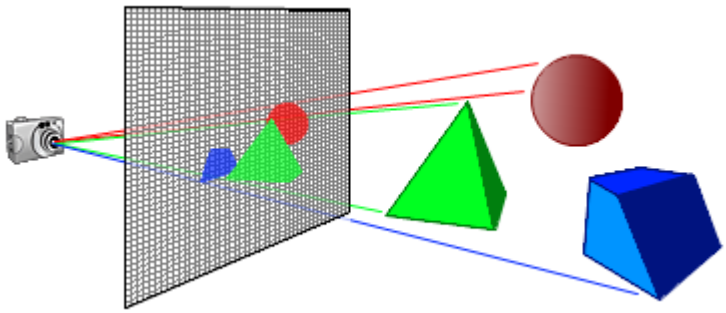

Figura 4.26: Ejemplo de proyección del raster sobre los objetos.

Se puede dar el caso en que el mismo punto de la malla tenga correspondencia con pixeles de dos o más rasters y para este tipo de casos el algoritmo se queda con el que el ángulo formado entre la normal del plano y la normal del punto del polígono tenga el menor grado posible.

Otro problema que puede suceder a la hora de aplicar esta técnica es que al seleccionar la textura para dos polígonos adyacentes, esta puede proceder de dos rasteres/cámaras diferentes, con sus respectivas variaciones de iluminación, colores, etc… y el algoritmo tiene la opción de tener esto en cuenta y producir un acomodamiento de estos cambios en forma de degradado.

4.4.2 Testeo sintético del algoritmo

En primera instancia, con la idea de probar el algoritmo he elegido un modelo de un avión, el cual se compone de una malla con su textura tal y como se puede ver en la siguiente imagen (ver Figura 4.27)

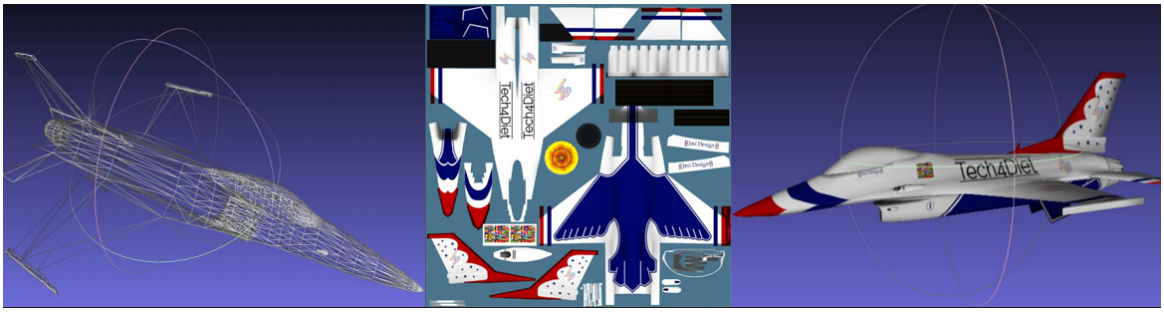

Figura 4.27: Modelo de malla mas textura empleado para el testeo.

Para realizar la prueba he empleado una función que tiene el programa Meshlab la cual me permite generar capturas con su raster y coordenadas x,y,z, como si de una cámara ubicada en el sistema de coordenadas del objeto se tratase. En concreto he realizado 14 capturas tal y como se puede ver en la siguiente imagen, (ver Figura 4.28) en la que podemos diferenciar el objeto en el centro y las 14 cámaras repartidas por el espacio global en función del enfoque desde el cual he realizado la captura.

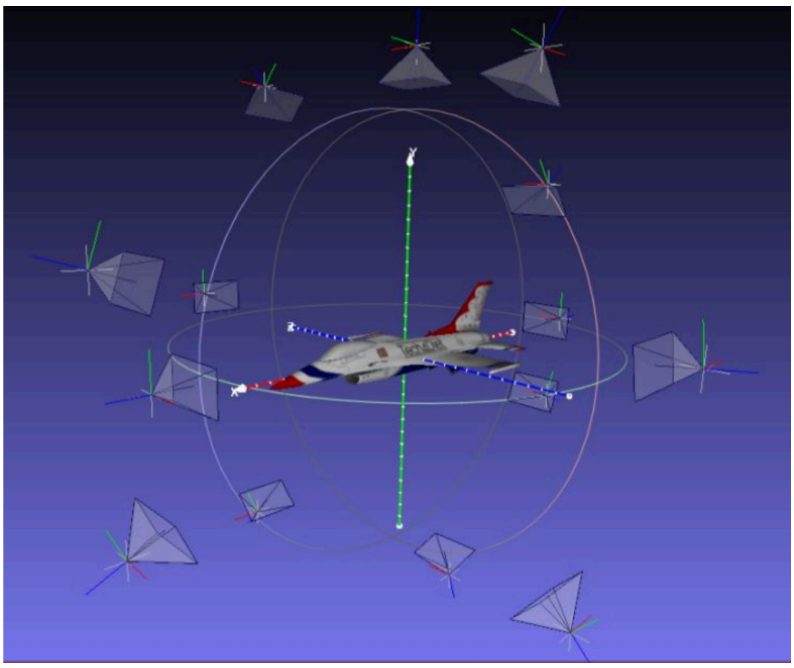

Figura 4.28: Ubicación de las distintas cámaras en el espacio global.

Por cada una de las capturas tengo el raster y los parámetros que obtendría del calibrado correspondientes a la cámara en cuestión. En la siguiente imagen se pueden observar estos datos correspondientes a una captura realizada desde la vista superior (ver Figura 4.29). Los datos que figuran en formato XML corresponden al nombre de la captura, los datos de calibrado intrínseco de la cámara, los datos de calibrado extrínseco y la ruta del raster en formato png.

```
<MLRaster label="F:/Administrador/Dropbox/T4D/textures/Transicion/recibido/e_top12.png">
 <VCGCamera FocalMm="25.3844" PixelSizeMm="0.0369161 0.0369161" ViewportPx="1200 794"
RotationMatrix="0.00711938 0.0110934 0.999913 0 0.999511 0.0303576 -0.00745371 0 -0.0304381 0.999478 -
0.0108722 0 0 0 0 1 " TranslationVector="0.0792582 -2.18489 -0.0433071 1" LensDistortion="0 0" CenterPx="600
397"/>
 <Plane semantic="1" fileName="../recibido/e_top12.png"/>
```

Figura 4.29: Información correspondiente a una de las vistas.

Para concluir el proceso ejecuto el filtro en Meshlab, al que le proporciono la resolución del mapa de textura de salida y genera un mapa de textura(ver Figura 4.30) con los segmentos de los diferentes rasters que ha empleado y la correspondencia de los pixeles de este con la malla del objeto en cuestión. En la siguiente imagen se puede observar el resultado de proyectar la nueva textura sobre la malla anterior (ver Figura 4.31).

Figura 4.30: Mapa de textura generado.

Figura 4.31: Modelo con la textura proyectada.

El resultado del mapa de textura y la correspondencia lo he guardado como un archivo de tipo *obj* que incluye los vértices, las caras y las correspondencias con el archivo de textura. De forma paralela guarda dos archivos a los que referencia, un archivo en formato *png* con la textura y otro archivo *mtl* que indica las cualidades del material a renderizar para mostrar el modelo. En las siguientes imágenes se puede observar el resultado obtenido con el sistema desarrollado (ver Figura 4.32 y Figura 4.33) así como el mapa de textura obtenido (ver Figura 4.34).

Figura 4.32: Vista frontal.

Figura 4.33: Vista trasera.

Figura 4.34: Mapa de textura generado.

4.4.3 Implementación

Para la ejecución del filtro se ha implementado de forma integrada en el conjunto la generación de un proyecto de Meshlab (ver Anexo 1, apartado 7.1), en el que se

incluyen cada una de las cámaras con sus correspondientes ajustes de calibrado y la relación de estas con los rasteres capturados.

Posteriormente se genera un archivo *mlx* en el que se incluyen las acciones a llevar a cabo una vez abierto el proyecto. Este tipo de archivo se genera automáticamente (ver Anexo 1, apartado 7.2) y guarda los filtros y acciones ejecutados junto con todos sus parámetros y se puede ejecutar como un script de forma automática.

Para el filtro de proyección de texturas Meshlab requiere un archivo XML con toda la información de las cámaras, este, al igual que el proyecto de Meshlab también se genera en base a los ajustes de calibrado y la cantidad de cámaras que haya conectadas en el sistema de forma automática. El código empleado se puede ver en el documento anexo (Ver Anexo 1, apartado 7.3)

Finalmente se realiza una llamada a través de línea de comandos al servidor de Meshlab que ejecuta los pasos necesarios establecidos en un script y posteriormente guarda el resultado en un archivo *obj.*

Todo este proceso se lleva a cabo de forma automática realizando la llamada por terminal a Meshlab server con el siguiente comando y parámetros:

```
LC_ALL=C snap run meshlab.meshlabserver -d FunctionsDebug.txt -l FilterDebug.txt -p
project.mlp -o Model.obj -m vc vn fc wt -s texture_projection.mlx
```

El parámetro LC_ALL=C se incluyó con el objetivo de arancar el servidor forzando a que emplee comas en los números decimales, puesto que esto estaba generando problemas de compatibilidad con Unity en el momento de la visualización del modelo exportado.

4.5 Integración del conjunto

A lo largo de esta sección se detalla cómo se ha coordinado la integración de todas las funciones dentro del programa, en primer lugar se habla de los pasos seguidos para

el diseño del diagrama de clases y su resultado, a continuación de la gestión de la carga de los distintos parámetros desde los archivos externos y para finalizar de cómo se han gestionado las distintas funciones que conforman el programa desarrollado.

4.5.1 Diseño diagrama de clases

El lenguaje de programación con el que se va a trabajar en toda la parte del servidor va a ser C++. Este lenguaje es un lenguaje de programación híbrido y multiparadigma que nos ofrece la posibilidad de trabajar con él mediante diferentes formas de abstracción. Uno de los paradigmas más populares de las últimas décadas es el paradigma orientado a objetos, paradigma que nació para dar solución a la llamada crisis del software, que permite una alta escalabilidad y que nos proporciona un modelo de abstracción que razona con las técnicas que las personas utilizamos para resolver los problemas (Shaalan, 1995).

El diagrama de clases planificado está dividido en cinco clases según sus responsabilidades, que son: Lógica de negocio, Núcleo, Capa de Acceso a Datos, Comunicación y Adquisición. La relación básica entre ellas se puede ver en el siguiente esquema (ver Figura 4.35), y la ampliación de cada una de las clases con sus funciones se pueden consultar en detalle en el Anexo 2

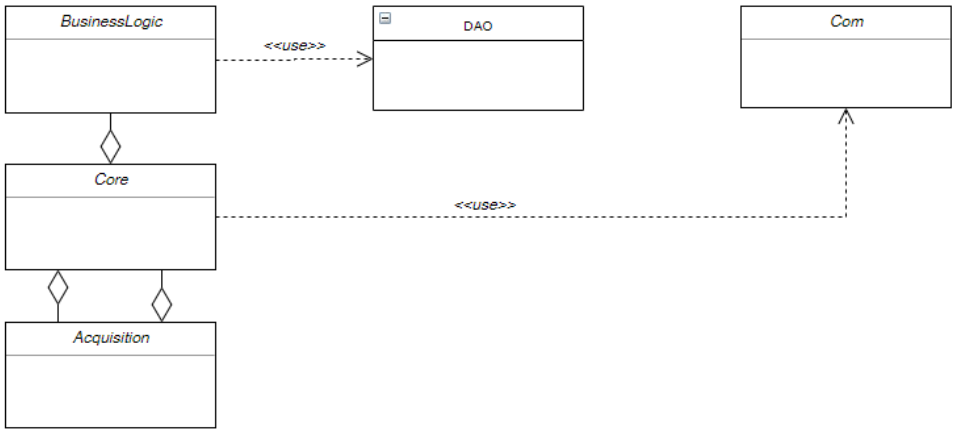

Figura 4.35: Esquema diagrama de clases.

4.5.2 Carga de archivos de configuración

La función “config”, dentro de la clase “Core”, implementa la tokenización de los archivos de configuración y carga todos los parámetros a sus respectivas variables en función de los valores incluidos en los archivos params.cfg y calibration.cfg. El primero incluye todos los ajustes de los filtros y el segundo todos los parámetros obtenidos de la calibración.

4.5.3 Gestión de modos de ejecución

Como modos de ejecución de la aplicación que se ejecuta en el servidor hay cuatro opciones que serán detalladas en este capítulo:

```
scanServer session_folder [-debug]
scanServer session_folder [-debug] -calib NumberCaptures CountDown
scanServer -testParameters inputFolder outputFolder ConfigFile
scanServer -serve
```

4.5.3.1 Sesión manual

Para realizar una captura de forma manual desde el servidor, como si de una sesión se tratase hay que indicar el directorio de la sesión donde se guardará la captura y opcionalmente el flag -debug, que guardará las nubes de puntos de los pasos intermedios a los filtros aplicados.

4.5.3.2 Calibrado

Para realizar una captura múltiple con la idea de usarla para llevar a cabo un calibrado, se puede usar el flag –calib seguido del número de capturas a realizar y el tiempo de cuenta atrás entre capturas. Esta opción se ha detallado con más profundidad en el capítulo 3 en el apartado de automatización del calibrado.

4.5.3.3 Test-Parameters

El flag -test-parameters, seguido de un directorio origen, otro de salida y un archivo de configuración, se ha implementado para facilitar el testeo de diferentes parámetros a la hora de ajustar los filtros sin tener que llevar a cabo una captura. El programa

cargará las nubes de puntos e imágenes desde la carpeta origen y generará el modelo mallado teniendo en cuenta los parámetros facilitados en el archivo de configuración.

4.5.3.4 Serve

El flag –serve deja el servidor a la escucha de las peticiones que pueda hacer el personal desde la parte de visualización. En la petición recibirá el directorio junto con los datos del paciente para crear su carpeta, sincronizar los pasos de la captura y posteriormente guardar la sesión escaneada en el mismo. Una vez terminada la captura queda a la espera de una nueva petición.

5 CONCLUSIONES

A lo largo de este capítulo detallaré las conclusiones obtenidas una vez finalizado este trabajo, comentaré líneas futuras que quedan abiertas con la posibilidad de mejorar, optimizar y avanzar en el desarrollo del sistema y finalmente expondré unas conclusiones personales acerca de la experiencia en torno al trabajo realizado.

5.1 Conclusiones

El trabajo desarrollado en este proyecto responde a la necesidad de obtener modelos 3D del cuerpo humano empleando técnicas de visión 3D y su representación texturizada. Estos modelos serán empleados para mejorar la adherencia a tratamientos en pacientes de obesidad mediante su visualización 3D y realidad virtual. En consecuencia, se ha desarrollado un sistema de adquisición de cuerpos 3D mediante redes de cámaras RGB-D para su uso en clínicas dietéticas que está previsto utilizar en centros de salud en el marco del proyecto de investigación en el que se ha desarrollado el trabajo.

De forma más concreta el trabajo fin de grado ha abordado una serie de objetivos específicos que fueron planificados en tareas y que han culminado con los siguientes logros:

Se ha diseñado e implementado un sistema de adquisición completo. Esto incluye la elección del hardware y el software necesarios, el dimensionado de la estructura de

red de cámaras y el diseño e implementación de la aplicación de captura de las distintas vistas (nubes de puntos) desde varios sensores RGBD.

Se han estudiado sistemas de calibrado intrínseco y extrínseco mediante los cuales se obtienen los parámetros de las cámaras y transformaciones necesarias para el posterior registro de las vistas adquiridas.

Se han analizado e implementado diferentes métodos de filtrado de las nubes de puntos correspondientes a cada vista para la mejora de la calidad de lo datos proporcionados por los sensores RGB-D.

Se han estudiado e implementado métodos de registro para la obtención de la nube de puntos del cuerpo completo a partir de las nubes correspondientes a las distintas vistas.

Se han estudiado métodos de obtención del modelo 3D a partir de la nube de puntos, en particular de mallado. En concreto se ha utilizado el método de Poisson para la obtención del modelo de malla del cuerpo a partir de la nube registrada, alcanzado unos resultados de representación acordes con los requerimientos de la aplicación.

Se ha generado el mapa de textura en base a las distintas imágenes raster correspondientes a las distintas vistas. Para ello se ha empleado un método Meshlab de correspondencia entre polígonos y vistas raster que optimiza la distribución de color de polígonos adyacentes, buscando coherencia en la textura obtenida. El resultado se exporta en formato estándar (OBJ) adecuado para su representación en el sistema de visualización 3D y realidad virtual.

Finalmente, todo este proceso se controla desde el interfaz de usuario de la aplicación, estando preparada para su integración con el subsistema de visualización.

5.2 Líneas futuras

Dentro del trabajo llevado a cabo hay diversas tareas susceptibles de ampliar, perfeccionar o investigar en un futuro, las cuales detallo a continuación:

- Dentro del apartado de adquisición, en un futuro se pretende perfeccionar el sistema para obtener el modelo del cuerpo partiendo de imágenes procedentes de una sola cámara, lo que supone abordar el problema del registro articulado.
- El calibrado actual ha proporcionado un resultado aceptable para una primera versión. En este ámbito está previsto analizar diferentes métodos de calibrado basados en diferentes marcadores 3D (cubo, esfera, …) que consigan optimizar la relación entre el número de vistas necesarias para el calibrado y la calidad conseguida.
- El calibrado es un proceso lento que es necesario optimizar. Teniendo en cuenta que el sistema estará en zonas transitadas se hace necesario que sea capaz de auto-diagnosticar su estado de calibración, llevando a cabo un calibrado de forma más automatizada que el sistema actual.
- En un futuro sería interesante perfeccionar la calidad del modelo obtenido, tanto la definición de la malla para poder obtener medidas más exactas, como la calidad de la textura para obtener resultados más realistas e inmersivos.
- Con la idea de optimizar el tiempo de generación del resultado una vez se ha escaneado se pretende incorporar una tarjeta gráfica al sistema para poder paralelizar muchas de las tareas mediante CUDA.
- Actualmente el proceso de ubicación en la cabina y de la pose del cuerpo para realizar la captura se realiza de forma visual, sería interesante

proporcionar un asistente automático de la pose con el fin de mejorar las comparaciones entre las diferentes modelos temporales.

5.3 Conclusiones personales

Todo el proceso de desarrollo de este trabajo ha sido altamente satisfactorio. He aprendido gran cantidad de conceptos que desconocía totalmente. En este espacio de tiempo tan reducido, el crecimiento personal y de conocimientos dentro del campo de la visión artificial ha sido muy significativo.

También he tenido la posibilidad de dar solidez y consistencia a muchos de los procesos, habilidades y aptitudes obtenidas a lo largo de los últimos cuatro años en los que he cursado el Grado.

Cuando te aproximas al último año de carrera los profesores suelen comentarte que, aunque no lo creas, estás mucho mejor preparado para afrontar el mundo exterior/laboral de lo que crees y que has adquirido unas capacidades que solo conocerás cuando las emplees.

Después de esta experiencia creo que he sido capaz de ampliar y reafirmar mis capacidades de abordar y llevar a cabo un proyecto de forma íntegra. Aunque suelo ser una persona valiente, reconozco que en un origen este proyecto me impuso respeto, pero aplicando lo aprendido durante estos años, creo que he sido capaz de alcanzar los objetivos planteados.

6 BIBLIOGRAFÍA

Azorín López, J. (2008). *Modelado de sistemas para visión de objetos especulares inspección visual automática en producción industrial*. Retrieved from https://rua.ua.es/dspace/handle/10045/7751

Bannai, N., Fisher, R. B., & Agathos, A. (2006). *Multiple color texture map fusion for 3D models*. https://doi.org/10.1016/j.patrec.2006.07.013

Callieri, M., Cignoni, P., Corsini, M., & Scopigno, R. (2008). Masked photo blending: Mapping dense photographic data set on high-resolution sampled 3D models. *Computers & Graphics*, *32*(4), 464–473. https://doi.org/10.1016/j.cag.2008.05.004

Fuster Guilló, A. (2004). *Modelado de sistemas para visión realista en condiciones adversas y escenas sin estructura*. Retrieved from https://rua.ua.es/dspace/handle/10045/9910

Ghosh, S. K. (2005). *Fundamentals of computational photogrammetry*. Concept Publishing Co.

Grunnet-Jepsen, A., Winer, P., Takagi, A., Sweetser, J., Zhao, K., Khuong, T., … Woodfill, J. (n.d.). *Using the RealSense D4xx Depth Sensors in Multi-Camera Configurations*. Retrieved from https://realsense.intel.com/wp-content/uploads/sites/63/Multiple_Camera_WhitePaper04.pdf

Han, X.-F., Jin, J. S., Wang, M.-J., Jiang, W., Gao, L., & Xiao, L. (2017). A review of algorithms for filtering the 3D point cloud. *Signal Processing: Image Communication*, *57*, 103–112. https://doi.org/10.1016/j.image.2017.05.009

Intel ® RealSense™ Depth Module D400 Series Custom Calibration. (2019). Retrieved from http://www.intel.com/design/literature.htm.

IOI Tech Corporation. (n.d.). Machine Vision USB3 | USB 3.0 PCIe Card | Quad Channel 4-port USB 3.0 PCI Express Card | PCIe x4 Gen II | U3X4-PCIE4XE111. Retrieved May 17, 2019, from http://www.ioi.com.tw/products/proddetail.aspx?CatID=106&DeviceID=3035&HostID=2035&ProdID=1060179

Kobbelt, L. P., Botsch, M., Schwanecke, U., & Seidel, H.-P. (n.d.). *Feature Sensitive*

Surface Extraction from Volume Data. Retrieved from https://www.graphics.rwth-aachen.de/media/papers/feature1.pdf

Liu, J., Bai, D., & Chen, L. (n.d.). *3-D Point Cloud Registration Algorithm Based on Greedy Projection Triangulation*. https://doi.org/10.3390/app8101776

Minghao Ruan, & Huber, D. (2014). Calibration of 3D Sensors Using a Spherical Target. *2014 2nd International Conference on 3D Vision*, 187–193. https://doi.org/10.1109/3DV.2014.100

Naveen, A., & Bandaru, N. (2016). Obstacle detection using stereo vision for self-driving cars. *Stanford*.

PCL Team. (2013a). Point Cloud Library (PCL): pcl::MedianFilter< PointT > Class Template Reference. Retrieved May 27, 2019, from http://docs.pointclouds.org/1.7.1/classpcl_1_1_median_filter.html

PCL Team. (2013b). Point Cloud Library (PCL): pcl::StatisticalOutlierRemoval< PointT > Class Template Reference. Retrieved May 27, 2019, from http://docs.pointclouds.org/1.7.1/classpcl_1_1_statistical_outlier_removal.html

PCL Team. (2019a). Point Cloud Library (PCL): pcl::BilateralFilter< PointT > Class Template Reference. Retrieved May 27, 2019, from http://docs.pointclouds.org/trunk/classpcl_1_1_bilateral_filter.html

PCL Team. (2019b). Point Cloud Library (PCL): pcl::GreedyProjectionTriangulation< PointInT > Class Template Reference. Retrieved May 27, 2019, from http://docs.pointclouds.org/trunk/classpcl_1_1_greedy_projection_triangulation.html

PCL Team. (2019c). Point Cloud Library (PCL): pcl::MarchingCubes< PointNT > Class Template Reference. Retrieved May 27, 2019, from http://docs.pointclouds.org/trunk/classpcl_1_1_marching_cubes.html

PCL Team. (2019d). Point Cloud Library (PCL): pcl::Poisson< PointNT > Class Template Reference. Retrieved May 27, 2019, from http://docs.pointclouds.org/trunk/classpcl_1_1_poisson.html

Radu Bogdan Rusu. (n.d.). Documentation - Point Cloud Library (PCL). Retrieved May 19, 2019, from http://pointclouds.org/documentation/tutorials/normal_estimation.php

Salman, N., Yvinec, M., Mérigot, Q., & Merigot, Q. (2010). Feature Preserving Mesh Generation from 3D Point Clouds. In *Computer Graphics Forum* (Vol. 29). Retrieved from https://hal.inria.fr/inria-00497632

Saval-Calvo, M. (2015). *Methodology based on registration techniques for representing subjects and their deformations acquired from general purpose 3D sensors*. Retrieved from https://rua.ua.es/dspace/handle/10045/49990

Saval-Calvo, M., Azorin-Lopez, J., Fuster-Guillo, A., & Garcia-Rodriguez, J. (2015). Three-dimensional planar model estimation using multi-constraint knowledge based on k-

means and RANSAC. *Applied Soft Computing*, *34*, 572–586. https://doi.org/10.1016/j.asoc.2015.05.007

Schwaber, K., & Sutherland, J. (2016). La guía de Scrum. *Scrum.Org*.

Shaalan, K. (1995). Software Development Environment Based on Object-Oriented and Logic Programming Paradigms. In *Object-oriented programmin with Java: Essential and Applications*. Retrieved from https://sites.google.com/site/khaledshaalan/publications/full-publication-list

Villena Martínez, V. (2015). *Análisis comparativo de métodos de calibrado para sensores RGB-D y su influencia en el registro de múltiples vistas*. Retrieved from https://rua.ua.es/dspace/handle/10045/48745

Zollhöfer, M., Stotko, P., Görlitz, A., Theobalt, C., Nießner, M., Klein, R., & Kolb, A. (2018). *State of the Art on 3D Reconstruction with RGB-D Cameras* (Vol. 37). Retrieved from http://onlinelibrary.wiley.com.

7 ANEXO 1

7.1 Generación proyecto de Meshlab:

```
void CloudGest::GenerateProjectMeshlab(){
  std::string filename = cfg.scanFolder + session_folder + "/project.mlp";
  std::cout << "Generating meshlab project: " << filename << std::endl;
  ofstream fileC;
  fileC.open(filename);

  fileC << "<!DOCTYPE MeshLabDocument>\n";
  fileC << "<MeshLabProject>\n";
  fileC << "  <MeshGroup>\n";
  fileC << "   <MLMesh label=\"Mesh-1.ply\" filename=\"Mesh-1.ply\">\n";
  fileC << "   <MLMatrix44>\n1 0 0 0 \n0 1 0 0 \n0 0 1 0 \n0 0 0 1 \n</MLMatrix44>\n";
  fileC << "   </MLMesh>\n";
  fileC << "  </MeshGroup>\n\n";
  fileC << "  <RasterGroup>\n";

  for(int idCam=0; idCam<cfg.cams.size(); ++idCam){
         fileC << "\t<MLRaster label=\"" << cfg.cams[idCam] << "-Color-1-calibrated.png\">\n";
  fileC << "\t<VCGCamera TranslationVector=\"0 0 0 1\" RotationMatrix=\"1 0 0 0 0 1 0 0 0 0 1
    0 0 0 0 1 \" PixelSizeMm=\"0.003 0.003\" LensDistortion=\"0 0\" CenterPx=\"640 360\"
    ViewportPx=\"1280 720\" CameraType=\"0\" FocalMm=\"1.93\"/>\n";
  fileC << "\t<Plane semantic=\"1\" fileName=\"" << cfg.cams[idCam] << "-Color-1-
    calibrated.png\"/>\n";
         fileC << "\t</MLRaster>\n\n";
  }
  fileC << "  </RasterGroup>\n";
  fileC << " </MeshLabProject>\n";
  fileC.close();
}
```

7.2 Generación script de acciones Meshlab:

```
int CloudGest::TextureGenerateScriptMeshlab(){
std::string filename = cfg.scanFolder + session_folder + "/texture_projection.mlx";
char buff[FILENAME_MAX];
getcwd( buff, FILENAME_MAX );
std::string current_working_dir(buff);

ofstream fileC;
fileC.open(filename);
       fileC << "<!DOCTYPE FilterScript>\n";
fileC << "<FilterScript>\n";

fileC << "<filter name=\"Matrix: Set from translation/rotation/scale\">\n";
       fileC << "<Param value=\"0\" type=\"RichFloat\" tooltip=\"Translation factor on X axis\"
name=\"translationX\" description=\"X Translation\"/>\n";
       fileC << "<Param value=\"0\" type=\"RichFloat\" tooltip=\"Translation factor on Y axis\"
name=\"translationY\" description=\"Y Translation\"/>\n";
       fileC << "<Param value=\"1\" type=\"RichFloat\" tooltip=\"Translation factor on Z axis\"
name=\"translationZ\" description=\"Z Translation\"/>\n";
       fileC << "<Param value=\"0\" type=\"RichFloat\" tooltip=\"Rotation angle on X axis\"
name=\"rotationX\" description=\"X Rotation\"/>\n";
       fileC << "<Param value=\"0\" type=\"RichFloat\" tooltip=\"Rotation angle on Y axis\"
name=\"rotationY\" description=\"Y Rotation\"/>\n";
       fileC << "<Param value=\"0\" type=\"RichFloat\" tooltip=\"Rotation angle on Z axis\"
name=\"rotationZ\" description=\"Z Rotation\"/>\n";
       fileC << "<Param value=\"1\" type=\"RichFloat\" tooltip=\"Scaling factor on X axis\"
name=\"scaleX\" description=\"X Scale\"/>\n";
```

```
	fileC << "<Param value=\"1\" type=\"RichFloat\" tooltip=\"Scaling factor on Y axis\"
name=\"scaleY\" description=\"Y Scale\"/>\n";
	fileC << "<Param value=\"1\" type=\"RichFloat\" tooltip=\"Scaling factor on Z axis\"
name=\"scaleZ\" description=\"Z Scale\"/>\n";
	fileC << "<Param value=\"false\" type=\"RichBool\" tooltip=\"If selected, the new matrix will
be composed with the current one (matrix=new*old)\" name=\"compose\" description=\"Compose with
current\"/>\n";
	fileC << "<Param value=\"true\" type=\"RichBool\" tooltip=\"The transformation is explicitly
applied, and the vertex coordinates are actually changed\" name=\"Freeze\" description=\"Freeze
Matrix\"/>\n";
	fileC << "<Param value=\"false\" type=\"RichBool\" tooltip=\"If selected the filter will be
applied to all visible mesh layers\" name=\"allLayers\" description=\"Apply to all visible
Layers\"/>\n";
	fileC << "</filter>\n";

fileC << "<filter name=\"Import cameras for active rasters from file\">\n";
	fileC << "\t<Param tooltip=\"It's possible to import both Bundler .out and Agisoft .xml files.
In both cases, distortion parameters won't be imported. In the case of Agisoft, it's necessary to
undistort the images before exporting the xml file\" description=\"Choose the camera file to be
imported\" type=\"RichOpenFile\" ext_val0=\"All Project Files (*.out *.xml);;Bundler Output
(*.out);;Agisoft xml (*.xml)\" name=\"ImportFile\" exts_cardinality=\"1\" ";
	fileC << "value=\"" << current_working_dir << "/" << cfg.scanFolder << session_folder <<
"/cameras.xml\"/>\n";
	fileC << "</filter>\n";
	fileC << "<filter name=\"Parameterization + texturing from registered rasters\">\n";
	fileC << "\t<Param tooltip=\"Specifies the dimension of the generated texture\" ";
		fileC << "description=\"Texture size\" type=\"RichInt\" name=\"textureSize\"
value=\"2048\"/>\n";
	fileC << "\t<Param tooltip=\"Specifies the name of the file into which the texture image will
be saved\" ";
		fileC << "description=\"Texture name\" type=\"RichString\" name=\"textureName\"
value=\"texture.png\"/>\n";
	fileC << "\t<Param tooltip=\"If true, the final texture is corrected so as to ensure seamless
transitions\" ";
		fileC << "description=\"Color correction\" type=\"RichBool\" name=\"colorCorrection\"
value=\"true\"/>\n";
	fileC << "\t<Param tooltip=\"It is the radius (in pixel) of the kernel that is used to compute
the difference between corresponding texels in different rasters. Default is 1 that generate a 3x3
kernel. Highest values increase the robustness of the color correction process in the case of strong
image-to-geometry misalignments\" ";
		fileC << "description=\"Color correction filter\" type=\"RichInt\"
name=\"colorCorrectionFilterSize\" value=\"3\"/>\n";
	fileC << "\t<Param tooltip=\"Includes a weight accounting for the distance to the camera
during the computation of reference images\" ";
		fileC << "description=\"Use distance weight\" type=\"RichBool\"
name=\"useDistanceWeight\" value=\"true\"/>\n";
	fileC << "\t<Param tooltip=\"Includes a weight accounting for the distance to the image border
during the computation of reference images\" ";
		fileC << "description=\"Use image border weight\" type=\"RichBool\"
name=\"useImgBorderWeight\" value=\"true\"/>\n";
	fileC << "\t<Param tooltip=\"If true, alpha channel of the image is used as additional weight.
In this way it is possible to mask-out parts of the images that should not be projected on the mesh.
Please note this is not a transparency effect, but just influences the weigthing between different
images\" ";
		fileC << "description=\"Use image alpha weight\" type=\"RichBool\"
name=\"useAlphaWeight\" value=\"false\"/>\n";
	fileC << "\t<Param tooltip=\"Remove all patches compound of a single triangle by aggregating
them to adjacent patches\" ";
		fileC << "description=\"Clean isolated triangles\" type=\"RichBool\"
name=\"cleanIsolatedTriangles\" value=\"true\"/>\n";
	fileC << "\t<Param tooltip=\"If true, texture coordinates are stretched so as to cover the
full interval [0,1] for both directions\" ";
		fileC << "description=\"UV stretching\" type=\"RichBool\" name=\"stretchingAllowed\"
value=\"false\"/>\n";
	fileC << "\t<Param tooltip=\"Extra boundary to add to each patch before packing in texture
space (in pixels)\" ";
		fileC << "description=\"Texture gutter\" type=\"RichInt\" name=\"textureGutter\"
value=\"4\"/>\n";
	fileC << "</filter>\n";
```

```
fileC << "</FilterScript>\n";
       fileC.close();
       return 1;
 }
```

7.3 Generación XML configuración de cámaras

```
int CloudGest::TextureGenerateCameraConfiguration(){
std::string filename = cfg.scanFolder + session_folder + "/cameras.xml";
ofstream fileC;
fileC.open(filename);
       fileC << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
fileC << "<document version=\"1.2.0\">\n";
fileC << "<chunk>\n";
fileC << "\t<sensors>\n";
for(int idCam=0;idCam<cfg.cams.size();idCam++){
       fileC << "\t\t<sensor id=\"" << idCam << "\" label=\"unknown0\" type=\"frame\">\n";
       fileC << "\t\t\t<resolution width=\"1280\" height=\"720\"/>\n";
       fileC << "\t\t\t<property name=\"pixel_width\" value=\"1\"/>\n";
       fileC << "\t\t\t<property name=\"pixel_height\" value=\"1\"/>\n";
       fileC << "\t\t\t<property name=\"fixed\" value=\"false\"/>\n";
       fileC << "\t\t\t<calibration type=\"frame\" class=\"adjusted\">\n";
       fileC << "\t\t\t\t<resolution width=\"1280\" height=\"720\"/>\n";
       fileC << "\t\t\t\t<fx>923.695</fx>\n";
       fileC << "\t\t\t\t<fy>923.641</fy>\n";
       fileC << "\t\t\t\t<cx>645.382</cx>\n";
       fileC << "\t\t\t\t<cy>359.797</cy>\n";
       fileC << "\t\t\t\t<k1>0</k1>\n";
       fileC << "\t\t\t\t<k2>0</k2>\n";
       fileC << "\t\t\t\t<p1>0</p1>\n";
       fileC << "\t\t\t\t<p2>0</p2>\n";
       fileC << "\t\t\t</calibration>\n";
       fileC << "\t\t</sensor>\n";
    }
    fileC << "\t</sensors>\n";
    fileC << "\t<cameras>\n";
for(int idCam=0;idCam<cfg.cams.size();idCam++){
 fileC << "\t\t<camera id=\"" << idCam << "\" label=\"" << cfg.cams[idCam] << "-Color-1-
  calibrated.png\" sensor_id=\"" << idCam << "\" enabled=\"true\">\n";
 fileC << "\t\t\t<transform>";
 std::vector<float> temp = cfg.matrices[idCam];
 fileC << temp[0] << " ";
 fileC << temp[1] << " ";
 fileC << temp[2] << " ";
 fileC << temp[3] << " ";
 fileC << temp[4] << " ";
 fileC << temp[5] << " ";
 fileC << temp[6] << " ";
 fileC << temp[7] << " ";
 fileC << temp[8] << " ";
 fileC << temp[9] << " ";
 fileC << temp[10] << " ";
 fileC << temp[11] + 1 << " ";
 fileC << "0.0 0.0 0.0 1.0</transform>\n";
 fileC << "\t\t</camera>\n";
}
fileC << "\t</cameras>\n";
fileC << "</chunk>\n";
fileC << "</document>\n";
fileC.close();
return 1;
}
```

8 ANEXO 2

8.1 Diagrama de clases